MEELMO

User Manual

Meelmo (base product) · Web HMI · Data access and SDKs · SQL Server historian · Oracle historian · TimescaleDB historian · QuestDB historian · ClickHouse historian · Industrial integration · Models · Studio Assistant

Version 0.4.2-rc · 2 November 2026

Contents

About this manual

Introduction

- What Meelmo is
- The base product and the optional modules
- Roles
- Conventions used in this manual
- The demonstration plants

Getting started

- First run: setting up the node
- Signing in
- The main window
- The menus
- The status bar
- Opening a demonstration
- Projects
- The built-in user manual
- Keyboard shortcuts

Acquisition and I/O configuration

- How acquisition is organised
- The I/O Configuration window
- Connections
- Acquisition groups
- Tags
- Circular buffers
- Lineage: what a tag feeds
- Tag drift
- Starting and stopping acquisition
- Acquisition diagnostics and timing
- Worked example: adding a tag to the packaging line

Signals: the signal tree, virtual and computed signals

- The Signal Tree
- Signal names
- Units
- Virtual signals
- The packaging line's computed signals
- Worked example: a computed signal
- Digital and text signals

Trend graphs

- What a trend graph is for
- Putting signals on a graph
- The parts of a trend graph
- Per-trace appearance
- Axis layouts and strips
- Digital and text signals
- Zooming, panning and going back
- Cursors and measurement
- Statistics: the Watch panel
- Notes that are saved with the project
- Per-channel analysis
- Several graphs, docking and tiling
- Exporting and analysis presets
- Keyboard and mouse summary

Recording and playback

- The Recording menu
- Recording by hand
- Recording settings
- Triggered recording
- Browsing recordings
- Playback
- Timing evidence and the recording timing analysis
- The recorder service
- Watching a remote recorder
- Recording files (.opslog)

Alarms

- How alarms work
- Alarm rules: Acquisition ▸ Alarms…
- Severities
- Notifications
- The alarm list
- Acknowledging
- Shelving
- Alarm history
- Alarm performance: Acquisition ▸ Alarm performance…
- Worked example: the packaging line's alarm rules

The historian

- The historian and recordings are different things
- Where the historian runs
- Providers: where history is stored
- Data ▸ Historian: the configuration window
- Retention and rollups
- Compressing stored history
- When the store is unavailable
- Sizing the disk
- Reading history: Data ▸ Browse History
- Worked example: retention for the packaging line, then CaseWeight over the last hour
- Moving history between stores

The semantic model

- The primitives
- What plant this is: the asset model
- What each kind of asset is made of: the role catalogue
- Assigning kinds and roles to signals
- Detection rules: runs, events, states and advisories
- The site's own words: glossary, owners and saved searches
- Moving a semantic model between sites
- Runs from a production database: run sources
- How the semantic model is used
- Worked example: the Checkweigher as a kind of asset

Datasets

- Why datasets exist
- Where datasets live, and how the Desktop reaches them
- What a dataset definition contains
- Data ▸ Dataset Studio
- Running a dataset over a time range
- The demonstration datasets
- Worked example: a checkweigher dataset, step by step

Reports

- What a report is
- Opening the Report Studio
- The Report Studio window
- Saving
- Previewing
- Running a report and getting the file
- The demonstration reports
- Worked example: an 8-hour shift report for the packaging line
- Reports made from an investigation
- Limits
- When something is wrong

Analysis

- Which tool, from where you are standing
- Run comparison
- The Analysis Studio
- Recording Timing Analysis
- Worked example: investigating a slow night on the packaging line
- When something is wrong

Configuration and deployment

- Projects and the node
- Configuration sections
- Drafts, activation and revisions
- Deployment topologies
- Trusted clients
- Fleet
- Backup and restore
- The public address
- The recorder service
- What the installer sets up

Users, roles and security

- How identity works in Meelmo
- Signing in
- Users and roles
- Changing your own password
- Signing out, sessions and seats
- The audit trail
- Stored credentials and key rotation
- Certificates and TLS
- Service accounts

Licensing and modules

- Editions
- The licence
- The trial
- Help ▸ License

Modules
Administration ▸ Modules
Seats
When a licence or trial ends

Diagnostics and support

Acquisition diagnostics
The commissioning report
Node history: configuration revisions
The support bundle
Evidence to keep and to send
The Help menu
Logs and file locations
What to include in a support request

Installation, setup and the programs

Before you install
The Meelmo Setup wizard
A server with no screen
Answer files and unattended installation
Repair, change, upgrade and remove
The setup command line
The Windows installer
Linux: where things go
Containers
The Community Edge VM
Running the programs by hand
A redundant pair

Control loops

Declaring a loop
The data comes first
What is checked, and how
Shared cycles: one fault, many loops
Reading the screens
The schedule, and the indices it keeps
Over the API and the assistant
What it cannot tell you
A demonstration
References

The browser client: finding your way

The top bar
The command palette
The phone layout
The status bar
When a screen stops working
Your own settings

The browser operator HMI

Signing in
The shell: header, page rail and status bar
Navigating between pages
The demonstration packaging line, page by page
The live widgets: value, gauge, state and mimic
Live trends: scope, logger, multiplot and stacked
Cards and dialogs over the page
Alarms
History replay
Commands
What an operator may and may not do
Assignments: who sees which pages
When there is no page to show
Stations, kiosks, phones and installed apps
The keyboard on the operator screens

The HMI Studio

Who may use it
The project browser
Creating a page
The page editor
Binding signals and choosing an update rate
Thresholds
Visibility, navigation and interactions
Animation
Composites and templates
Strings
Theme
Navigation
Layout and breakpoints
Preview and simulate
Validation and page cost
Editing at the same time as somebody else
Publishing, versions and rollback
Assignments
Worked example: a case weight gauge on the Overview page
Worked example: drawing the case magazine and checkweigher infeed
Troubleshooting

Automation rules

Who may do what
The Automation workspace

- The rule editor
- The guided editor
- The graph editor
- The block catalogue
- Execution policy
- Testing a rule
- Activating, revisions and deleting
- Runs and traces
- Notifications
- Connectors
- Helpers
- Packages
- Repairs and setup
- The three demonstration rules
- Troubleshooting

Reports and analysis in the browser

- Who sees what
- The Reports area
- Running a report and taking it away
- The Report Studio
- The signal catalogue
- Lineage
- Analysis in the browser
- Ask the data
- What changed
- What relates
- Process capability
- Data quality
- Cross-check
- Compare places
- Alarm performance
- Process digest
- Analytics as your start page
- Compare runs

Administration in the browser

- Reaching the administration screens
- Users & permissions
- Service accounts
- Stations
- Audit trail
- Certificates
- I/O
- Tag drift
- Lineage
- Historian
- Run sources
- Linked nodes
- Fleet
- Single sign-on
- Branding
- License
- Modules
- Studio Assistant
- Interface primitives
- First-run setup (`/setup`)

HMI component reference

- How to read an entry
- Choosing a component
- Containers
- Values and KPIs
- Gauges and indicators
- Trends and analytics
- Alarms and events
- Process
- Shapes and symbols
- The symbol library
- Execution and context
- Navigation
- Information
- Worked example: the four charts on the Trends page
- Worked example: the control-room panels on the Overview and Machines pages
- Worked example: machines and conveyors on the Line mimic page
- Worked example: a line account on the OEE page

HMI assets, expressions, parameters and keys

- Assets: images and 3D models
- Expressions
- Page parameters
- The tabs component
- Conditions and colours in one place
- Moving a project: export and import
- Keys in the Studio
- Keys in the operator screens

Data access and SDKs

- The data-access server
- Service accounts
- The REST API
- Worked example: fetching a dataset with curl
- The Python SDK
- The .NET SDK

Grafana
Deploying integrations safely
MCP from a desktop assistant (standard input and output)
When it does not work

SQL Server historian
What this module adds
Before you start
Choosing SQL Server as the store
Connection fields
Test, Validate and Create schema
Worked example: moving the packaging line to SQL Server
Retention and rollups
Ingestion and the spool
Read alongside and migration from the embedded historian
Reporting views
Sizing
Troubleshooting

Oracle historian
What this module adds
Before you start
Choosing Oracle as the store
Connection fields
Test, Validate and Create schema
Worked example: the packaging line on Oracle
Retention and rollups
Ingestion and the spool
Read alongside and migration from the embedded historian
Reporting views
Sizing
Troubleshooting

TimescaleDB historian
What this module adds
Before you start
Choosing TimescaleDB as the store
Connection fields
Test, Validate and Create schema
Worked example: the packaging line on TimescaleDB, read by Grafana
Retention and rollups
Ingestion and the spool
Read alongside and migration from the embedded historian
Reporting views
Sizing
Troubleshooting

QuestDB historian
What this module adds
What it does not hold
Before you start
Choosing QuestDB as the store
Connection fields
Test, Validate and Create schema
Written, then readable
Retention and rollups
Ingestion and the spool
Migration
Sizing
Troubleshooting

ClickHouse historian
What this module adds
What it does not hold
Before you start
Choosing ClickHouse as the store
Connection fields
Test, Validate and Create schema
Stored once, eventually; read once, always
Retention and rollups
Ingestion and the spool
Migration
Sizing
Troubleshooting

Industrial integration: MQTT, Sparkplug and MCP
What this module contains
Publishing samples onward
The MQTT transport
The Sparkplug B transport
The OPC UA server
Kafka
The MCP surface for AI clients
The node's own MQTT broker
Connectors are not part of this module
Troubleshooting

Models

What models are for
Where to find it
The Models Studio on the Desktop
The Models page in the browser
Estimating what is not measured: step by step
Worked example: estimating case weight on the packaging line
Watching what the model gets wrong
Forecasting: How far ahead
Keeping a model
Running a trained model at the edge
Where an assistant fits
Troubleshooting

The Studio Assistant

What it does
What must be true before it answers
Settings in the browser
Shortlist advice on the Desktop
Asking it in the HMI Studio
What it can and cannot change
Approvals, risk and auto-apply
Privacy: what is sent, and where
Limits
Troubleshooting

Troubleshooting

Where to look first
It will not start
Nobody can sign in
It will not connect
The data is not what you expected
The historian is unavailable
Something is unavailable
Something is slow
Recovering
Where logs live
What to send with a report

About this manual

This manual describes Meelmo 0.4.2-rc as delivered to your site. It is arranged in chapters, one group for each part of the product, and it contains the chapters for the modules your installation is licensed for:

- **Meelmo (base product)** — included with every installation
- **Web HMI**
- **Data access and SDKs**
- **SQL Server historian**
- **Oracle historian**
- **TimescaleDB historian**
- **QuestDB historian**
- **ClickHouse historian**
- **Industrial integration**
- **Models**
- **Studio Assistant**

This copy covers every module.

Figures show the demonstration packaging line that ships with Meelmo, so every screen is shown with real signals, alarms, reports and rules on it. Your own plant's names appear in their place.

Introduction

This manual describes Meelmo release `0.4.2-rc`: what each part of the product is for, every screen an engineer, operator or administrator works with, and how to use them. It is arranged in chapters, one set per licensed module, so that the copy delivered to a site contains only the parts that site has bought. This chapter explains what Meelmo is, how it is divided, who may do what, and the conventions the rest of the manual uses.

What Meelmo is

Meelmo is an industrial data platform. It acquires signals from controllers, computes derived signals from them, shows them live, records them, stores them long-term in a historian, raises alarms on them, and serves them to operators and to other systems.

It is built from a small number of parts, and understanding which part does what makes the rest of the manual easier to follow.

The node

A **node** is the authority everything else acts through. It is one acquisition project with one owner, and it has its own user accounts, its own configuration revisions and its own tamper-evident audit trail. **One process hosts a node; every other process attaches to it as a client.** A controller is therefore polled once, however many Desktops and browsers are watching it.

The process that hosts a node is sometimes called the *agent* or the *recorder service*. On a workstation it runs inside the Desktop application; on a plant server it runs as an unattended service that starts with the machine, runs without anybody signed in and survives a reboot.

A node owns:

- the **acquisition**: the connections to controllers, their acquisition groups and tags, and the drivers that read them;
- the **computed signals**, **alarm rules** and **detection rules** evaluated over what is acquired;
- the **historian** configuration and the history recorded into it;
- the **accounts**, **roles** and **audit trail**;
- the node's **configuration revisions**, so that every change is recorded with who made it and can be rolled back.

The Desktop engineering client

The **Desktop** is the engineering and commissioning client. It runs on Windows, macOS and Linux. In the Desktop you:

- describe the plant: connections, drivers, acquisition groups, tags and circular buffers;
- define virtual (computed) signals;
- start and stop acquisition and check its timing;
- plot live and recorded signals on trend graphs, measure them with cursors and analyse them;
- record to files and play recordings back;
- configure alarms, detection rules and the historian, and browse history;
- build datasets and reports, and analyse what changed around an event;
- administer the node: users, service accounts, deployment, modules, certificates, backup and diagnostics.

The Desktop has no operator "run the plant" runtime and no operator faceplates. Operator screens are served to the browser.

The historian

The **historian** is the node's long-term store. Where a recording is a file made between a start and a stop, the historian is a database with retention and rollups that answers questions over months. The base product includes an embedded historian and a SQLite historian; SQL Server, Oracle, TimescaleDB, QuestDB and ClickHouse historians are optional modules. The historian is configured in the Desktop under **Data ▸ Historian**, and read by the history browser, datasets, reports and analysis.

The browser HMI

The **browser HMI** is the operator surface: operators sign in with a web browser to watch the live plant, read trends, work alarms and replay history, on pages that engineers author in the browser's HMI Studio. Nothing is installed on an operator workstation. The browser HMI is the **Web HMI** module; an installation that has not bought it has no browser surface, and everything in the base chapters of this manual is done from the Desktop.

The base product and the optional modules

Everything needed to acquire, look at, record, store and administer a plant is the **base product**. The Modules window in the product lists it beside the modules so it is clear which half a site has bought:

Base product	
Desktop engineering client	The application this manual's base chapters describe
Embedded agent	The node hosted inside the Desktop, or as a service on a plant server
Acquisition and the standard protocol drivers	Every driver listed in the *Acquisition and I/O configuration* chapter
Simulator	A driver that generates waveforms per tag, so a project runs with no hardware
Trends, alarms and recording	Live and recorded trend graphs, alarm rules, recording to files
SQLite historian	Long-term history on this machine (the embedded store is also included)
Configuration, RBAC, backup and restore	Configuration revisions, role-based access control, node backup
Commissioning report	The evidence document for handing a site over
Semantic rules, datasets and reports	Detection rules, the semantic model, saved queries and reports

The **optional modules** are licensed separately. A module a site buys later arrives as a signed file, is installed from **Administration ▸ Modules**, and arrives switched off: turning it on is a separate, recorded decision.

Module	**What it adds**
Web HMI (`web-hmi`)	The browser operator runtime and the host that serves it: operator pages, live widgets, the alarm drawer, history replay, gated commands, and in the browser the HMI Studio, Report Studio, automation rules and administration pages
Data access and SDKs (`data-access-sdk`)	The data-access server, its REST and gRPC surfaces, and the Python and .NET SDKs that read them; the route by which other systems (Grafana, a data science notebook, a MES) read history, live values, datasets and the semantic model

Module	What it adds
SQL Server historian (`historian-sqlserver`)	Writing this node's history to Microsoft SQL Server
Oracle historian (`historian-oracle`)	Writing this node's history to Oracle
TimescaleDB historian (`historian-timescale`)	Writing this node's history to TimescaleDB (PostgreSQL)
QuestDB historian (`historian-questdb`)	Writing this node's history to QuestDB (samples and rollups; the semantic model stays in the journal)
ClickHouse historian (`historian-clickhouse`)	Writing this node's history to ClickHouse (samples and rollups; the semantic model stays in the journal)
Industrial integration (`industrial-integration`)	Sold together: outbound MQTT and Sparkplug B publication for a unified namespace, Kafka publication, the node's governed MQTT broker, the read-only OPC UA server, and the MCP server through which an external assistant can query the plant read-only
Models (`models`)	Fitting models to recorded history and estimating from them what the plant does not measure (soft sensors), with forecasts and model monitoring
Studio Assistant	An authoring assistant inside the browser's Report Studio, HMI Studio and automation editor that proposes changes from a description in plain words. It is licensed by a capability in the licence rather than switched on in the Modules window, and it also needs a configured model provider

Chapters that belong to a module open with a note saying so. If a module's chapters are missing from your copy of this manual, your installation is not licensed for that module.

Two further points about what an installation can run:

- **Editions.** Meelmo is delivered as Community, Standard and Professional downloads. Community is free, needs no licence file, is limited to 32 tags and ships only the Simulator, Modbus TCP/RTU, OPC UA, MQTT, MTConnect, IEC 60870-5-104, DNP3, BACnet/IP, Tasmota, Sparkplug B, Shelly, SunSpec, LoRaWAN, IO-Link, M-Bus and KNX/IP drivers. Standard (256 tags) and Professional (unlimited tags, continuous and triggered recording, the recorder service) carry every driver. The *Licensing and modules* chapter describes this in full.
- **The trial.** A paid edition with no licence installed runs for 30 days with every module enabled at small quantities.

Roles

Every action that changes configuration or commands a process is authorised against the signed-in account and written to the audit log. A person holds one of four roles:

Role	May do
Viewer	Read-only: open projects, look at live values and trends, play back recordings, browse history and export. No control and no configuration
Operator	Everything a Viewer may, plus operate the running system: start and stop acquisition, start and stop recording, and, where a connection and a tag have been made writable, write a value to a controller tag. An Operator cannot change configuration

Role	May do
Engineer	Everything an Operator may, plus configure: I/O configuration, virtual signals, alarms, the historian, saving the project, and installing and starting the recorder service
Administrator	Everything, including managing user accounts, installing a licence, modules, backup and key rotation

The roles form a ladder: each includes everything the one below it may do. Machine-to-machine callers (service accounts) do not hold a role; they are granted individual capabilities directly. On a node whose directory defines further named roles (for example a *Shift lead*), a person can hold those as well; the Desktop shows the ladder role and keeps the others intact when an administrator edits the account.

The accounts are managed under **Administration ▸ Users & Security**, described in the *Users and security* chapter.

Conventions used in this manual

Convention	Meaning
Data ▸ Historian	A menu path: the **Data** menu, then its **Historian…** entry. An ellipsis on a menu entry in the product means it opens a window
OK, **Test Connection**	A label, button, field or tab as it appears on screen
`Ctrl+S` , `F5`	A keyboard gesture
`Packaging/Checkweigher/CaseWeight`	A value, a file name or a signal identifier, typed as shown
`[Packaging/Checkweigher/CaseWeight]`	A signal reference inside an expression. The square brackets are part of the expression syntax

How a signal is named. Throughout the product a signal is identified by its *topic*, `Connection/Group/Tag`: the connection it is read through, the acquisition group it is read in, and the tag's own name. `Packaging/Checkweigher/CaseWeight` is the gross case weight read by the connection `Packaging` in the group `Checkweigher`. Computed signals have the topic `Virtual/Computed/<name>`, for example `Virtual/Computed/WeightDeviation`. Alarm rules, datasets, reports, HMI bindings and the data-access API all use the same names.

Screenshots. The figures were captured from the development builds of release `0.3.1` with the Desktop attached to a node running the packaging-line demonstration, in the dark theme and in English. A few figures come from earlier builds or from another demonstration plant; their captions say so. The product can switch to a light theme and to nine other languages, so your screens may differ in colour and wording but not in layout.

The demonstration plants

Meelmo ships five simulated plants. They are ordinary project files in ordinary product formats, fully configured: the tags and their acquisition rates, the computed signals, the alarms, the detection rules and the prepared charts are in the file. They run on the **Simulator** driver, so they need no controller. They are opened from **Help ▸ Demonstration…**, described in the *Getting started* chapter.

Plant	File	Enabled tags	What it shows
Packaging line	`packaging.meelmoproj`	118	A ten-machine case-packing line. Used for the examples throughout this manual

Plant	File	Enabled tags	What it shows
Galvanizing line	`galvanizing.meelmoproj`	35	A continuous galvanizing line, measured by the metre of strip
Meltshop	`meltshop.meelmoproj`	51	Production by heat rather than by metre
Hot strip mill	`hotstrip.meelmoproj`	59	A hot strip mill
Reference works	`reference.meelmoproj`	28	A simulated water treatment works whose job is to be a worked example of every configuration option the other four do not use: circular buffers, an OPC UA subscription, a scaled tag, a tag with a role, a C# computed signal, alarm delays, the notification channels, lossless recording, a role catalogue and an asset model. It carries thirteen connections, one per driver the product ships; only the Simulator connection is enabled

Each plant also has a matching set of operator screens (`packaging.meelmohmi` and so on), which are imported in the browser when the Web HMI module is licensed, and example automation rules.

The packaging line

The packaging line is the plant used in the examples in every chapter. It has one connection, `Packaging`, using the Simulator driver, with one acquisition group per machine and three for the line as a whole:

Group	Poll rate	What it is
`Erector`	500 ms	Erects a folded blank into an open case
`BottomSealer`	500 ms	Tapes and compresses the bottom flaps
`Loader`	500 ms	Collates pouches and top-loads them into the case. It sets the line rate
`Checkweigher`	500 ms	Weighs every case and signals the reject pusher
`TopSealer`	500 ms	Tapes and compresses the top flaps
`Labeler`	500 ms	Prints and applies the case label
`Vision`	500 ms	Reads the barcode and checks label placement
`Reject`	500 ms	Pushes a failed case into the chute
`Palletiser`	500 ms	Places cases onto the pallet in its stacking pattern
`Wrapper`	500 ms	Wraps a completed pallet and discharges it
`Conveyors`	200 ms	Accumulation buffer levels between machines, in %

Group	Poll rate	What it is
`Line`	1000 ms	Line counters and utilities: `GoodCount`, `RejectCount`, `InfeedCount`, `TargetRate`, `OrderTarget`, `AirPressure`, `ElectricalPower`, `OrderBatch`, `ProductCode`, `ChangeoverActive` and the reject counts by cause
`CasePacking`	200 ms	The case-packing cell, watched one case at a time: `CaseState`, `RobotState`, `CycleTimeSec`, `PalletCount`, `CasesPerHour` and others

Every machine group carries the same core tags — `Rate` and `RatedSpeed` (cases/min), and the four Boolean states `Running`, `Faulted`, `Blocked` and `Starved` — plus tags of its own, such as `Checkweigher/CaseWeight` and `Checkweigher/WeightTarget` (g), `Loader/PouchesPerCase`, or `Erector/MagazineLevel` (%).

On top of the acquired tags the project defines 28 computed signals — a state code and a throughput per machine, and line-level `Availability`, `Performance`, `Quality`, `OEE`, `WeightDeviation`, `RateDeviation`, `UnaccountedUnits` and `OrderProgress` — and seven alarm rules, among them *Pouch loader faulted*, *Case weight out of tolerance* and *Compressed air low*.

When the demonstration's saved queries and reports are installed, the node also has the packaging line's datasets and reports, used in the *Datasets* and *Reports* chapters.

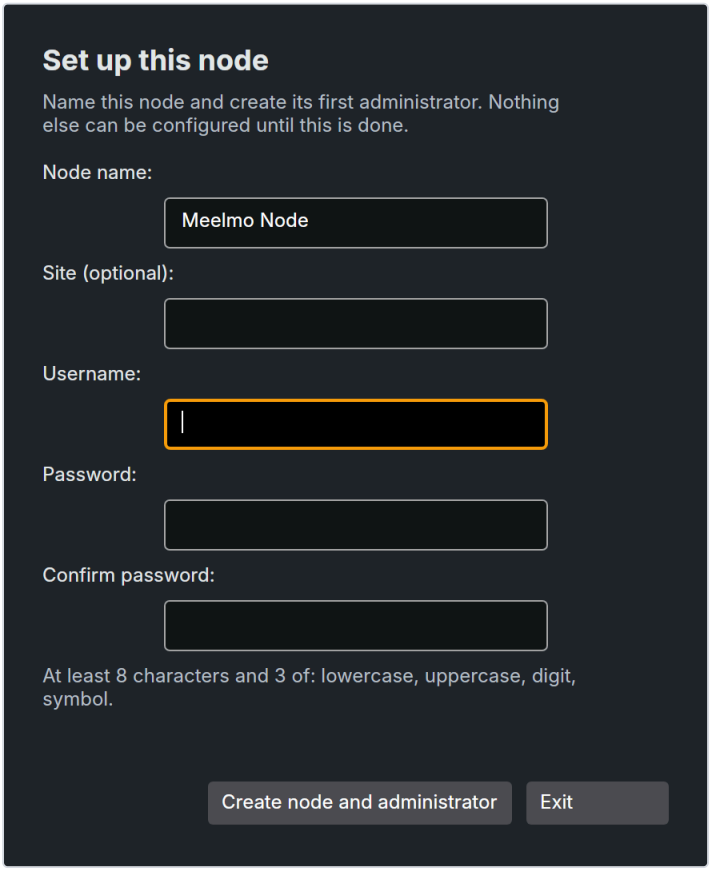
The screenshots were taken with two seeded accounts: `admin`, an Administrator, and `shift`, an Operator.

Getting started

This chapter takes you from the first launch of the Desktop to a running plant on your screen. It covers commissioning a new node, signing in, the parts of the main window, every menu, the status bar, opening a demonstration, working with projects, the built-in user manual and the keyboard shortcuts.

First run: setting up the node

The first time the Desktop is started on a machine whose node has never been commissioned, it shows **Set up this node** instead of a sign-in window. A node with no administrator cannot be configured, so this step cannot be skipped and there is no default password.



The Set up this node window shown on first launch, asking for a node name, an optional site, and the first administrator's username and password

Field	What to enter
Node name	The name this node is known by in the audit trail, in the Fleet window and in the browser. It is pre-filled with `Meelmo Node`; give it a name that identifies the machine or line, for example `Packaging Line 1`
Site (optional)	The plant or site the node belongs to, for example `Leeds`. Used to group nodes
Username	The first administrator's account name, for example `admin`
Password / **Confirm password**	At least 8 characters, using at least 3 of: lowercase, uppercase, digit, symbol

Press **Create node and administrator**. **Exit** closes the application without commissioning anything.

The account created here is the one the node's audit trail is anchored on. Every later account is created under **Administration ▸ Users & Security** by somebody who has signed in.

Two variants of this window exist:

- **A node on another machine.** When the Desktop is a client of a node that runs elsewhere and that node has never been commissioned, the window also asks for the **One-time claim** the node printed when it first started. This proves that the person commissioning it has access to the machine it runs on. If the claim can be read from the node's own directory on this machine, it is filled in for you.
- **A plant server.** A plant server installed with the installer is commissioned by the installer, which asked who administers it. It never shows this window.

A newly commissioned node starts with a small **example line** — one simulated endpoint, ten signals — so that it shows live data immediately. It is ordinary configuration: it appears in the I/O configuration and can be removed like anything else. Remove it before handing a site over; the commissioning report refuses a node whose only endpoint is the example line.

Signing in

On every later start, and after **Administration ▸ Sign out**, the Desktop shows **Sign in to Meelmo**.



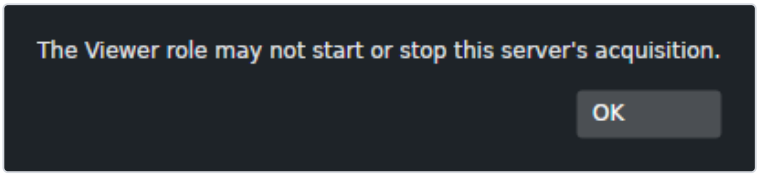
The Sign in to Meelmo window with Username and Password fields and the Sign in and Exit buttons

1. Type your **Username** and **Password**.
2. Press **Sign in** (or `Enter`).

What can go wrong:

Message	Meaning
Invalid username or password.	Either is wrong. The message deliberately does not say which
Too many failed attempts. Try again in N s.	The account is locked out for the time shown after repeated failures
Set a password for this account	The account exists but has no password yet (for example, one an administrator created without one). Enter a **New password** that meets the policy

What the signed-in account may do depends on its role (Viewer, Operator, Engineer or Administrator), described in the *Introduction*. Menu entries that need a permission the account does not hold remain visible, but the action is refused with a **Not permitted** message (*Your role (Viewer) is not permitted to: …*), and nothing is changed. Acting on the plant is checked by the node too: a Viewer who presses **Start** on a client workstation is told so by the server, and its acquisition keeps its state.

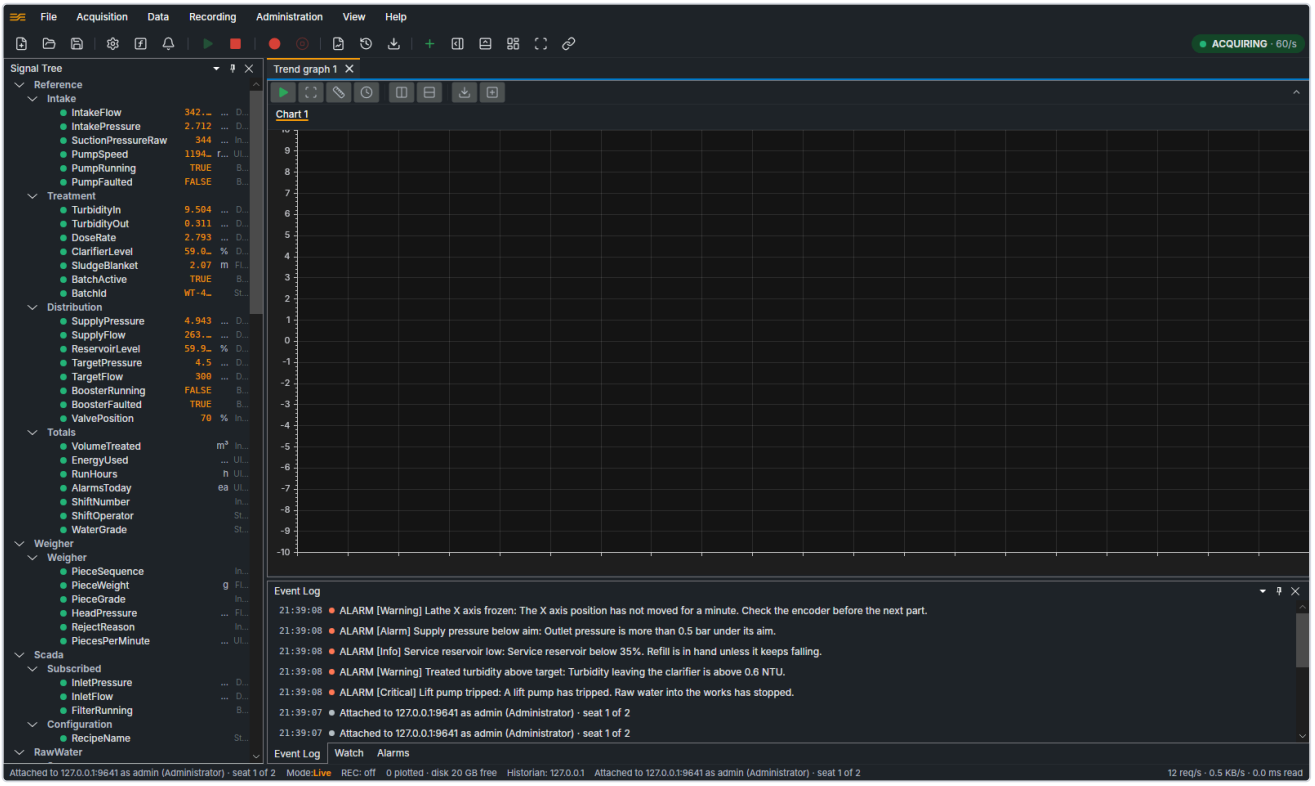


The message a Viewer sees on pressing Start: The Viewer role may not start or stop this server's acquisition, with an OK button

Leaving a shared workstation. Administration ▸ Sign out returns to the sign-in window without closing the application. **Administration ▸ Change my password…** changes your own password.

The main window

The main window is a menu bar, a toolbar, dockable panels, a document area holding the trend graphs, and a status bar.



The Desktop main window attached to a node running the packaging line: the Signal Tree on the left lists the Packaging connection with its Erector, BottomSealer, Loader, Checkweigher, TopSealer and Labeler groups; an empty Trend graph 1 fills the centre; the Event Log is below it; the ACQUIRING pill at top right reads 284/s

The figure shows the Desktop attached to a node running the packaging-line demonstration. From top to bottom:

1. **The menu bar**: **File**, **Acquisition**, **Data**, **Recording**, **Administration**, **View** and **Help**, headed by the Meelmo mark.
2. **The toolbar**: icon buttons in six groups, and on the right the acquisition and recording state.

3. **The Signal Tree** panel on the left, **Trend graph 1** in the document area, and the **Event Log**, **Watch** and **Alarms** panels below it.
4. **The status bar** along the bottom.

The toolbar

The toolbar carries icons only. Every button has a tooltip naming the action and its shortcut. The buttons are grouped by purpose, and a button whose action cannot be performed right now is disabled — Start while acquisition is already running, Stop while it is not, and the same for the two recording buttons.

Group	Buttons, left to right
Project	New project, Open project, Save project
Configure	I/O configuration, Virtual signals, Alarms
Operate	Start acquisition (`F5`), Stop acquisition (`F6`)
Record	Start recording, Stop recording
Data in and out	Open recording, Browse history, Export plotted signals to CSV
Charts	Add trend graph, New trend graph split right, New trend graph split down, Apply analysis (chart preset), Zoom to fit, Lock every trend graph to the same time axis (a toggle that stays lit while on)

The Operate group is separated from its neighbours by wider gaps, because Start and Stop are the two buttons that must not be pressed by accident. The shapes — a triangle for Start, a square for Stop, a filled circle for Record, a circle in a square for Stop recording — distinguish them without relying on colour.

On the right of the toolbar:

- **The acquisition pill** reads **STOPPED** or **ACQUIRING**, followed by the number of samples per second (`284/s` in the figure). If the acquisition loop cannot keep up with a requested poll rate, a warning `⚠ N overruns` is appended.
- **The REC pill** appears only while recording and shows the number of samples recorded.

The panels

Panel	What it is for
Signal Tree	Every connection, group and tag of the open project, with each tag's live value, unit and data type and a quality dot. Computed signals appear under their own **Virtual ▸ Computed** branch. Drag a tag onto a graph, or double-click it, to plot it. See the *Signals* chapter
Event Log	What the application did, newest first, with a time and a coloured severity dot (see *The Event Log* below)
Watch Window	One row per signal plotted on the active graph: its latest **Value**, its value under the measurement **Cursor**, and **Min**, **Max**, **Mean** and **σ** (standard deviation) over the visible window, with the **Unit**. See *Statistics: the Watch panel* in the *Trends* chapter
Alarms Panel	The alarms the node's rules are holding now, and those that have cleared

The Event Log

The Event Log is the running account of what this Desktop did and what it was told, so you can see why something happened without opening the log files. Each line has the local time (`HH:mm:ss`), a dot giving the kind of entry, and the message.

Dot	Kind	Typical entries
Grey	Information	*New project*, *Configuration loaded from node* '*...*', acquisition starting and stopping, a recording started, demonstration queries and reports installed, a newer release available
Green	Good news	A connection coming up, an alarm clearing
Amber	Warning	A connection reconnecting, an acknowledgement the node refused, a historian source that cannot be read, a gap in what the node delivered
Red	Problem	A connection lost or failing, with the driver's own reason; an alarm raised, with its severity and message; an alarm rule that cannot be evaluated; the licensed seat lost

Points to know:

- **Newest first.** A new entry goes to the top. The panel keeps the newest 500 entries; older ones drop off the bottom.
- **Status messages are copied in.** Every message the status bar shows is also written here, once, so a message that has since been replaced on the status bar can still be read.
- **Long messages are shortened** with an ellipsis to fit the panel. Hover over a line to read the whole message in its tooltip, or widen or float the panel.
- **It belongs to this session.** The panel has no clear, copy or save command, and starts empty each time the Desktop starts. For a lasting record use the node's audit trail (*Users and security*) or **Help ▸ Save Support Bundle…**, which collects the application's log files (*Diagnostics and support*).

When a connection will not come up, the Event Log is the first place to look: the red line names the connection and gives the reason the driver reported.

When a project has no signals yet, the Signal Tree shows **No signals yet** with a **Set up I/O…** button that opens the I/O configuration.

Each panel's title bar has three controls: a drop-down to **float** it into a window of its own, a pin to **auto-hide** it to a tab on the window edge, and × to close it. Closing or auto-hiding a panel gives its space to the trend graphs. A closed panel is brought back from the **View** menu. The Event Log, Watch and Alarms panels share one docking area and are switched with the tabs at its bottom.



The main window acquiring from the packaging node: the signal tree on the left with the plotted tags marked 1, a trend graph of CaseWeight, WeightTarget, the checkweigher rate, seal temperature, air pressure and electrical power in strips, the Event Log below, ACQUIRING and REC on the toolbar, and the status bar saying 6 plotted

The second figure shows the same window with six packaging signals plotted: in the Signal Tree they are drawn in bold with a small badge giving the number of the trend graph they are on, and each has its own strip on the graph with its latest value and range in the legend. A Boolean such as `Running` is drawn as a state band rather than a line.

Trend graphs

Each trend graph is a tab in the document area (**Trend graph 1**, **Trend graph 2**, …) with its own toolbar — Play, Fit, Measure, Time, stack horizontally, stack vertically, Export and new chart tab — and one or more chart tabs (**Chart 1**). The chevron at the right end of the chart toolbar collapses it. Graphs can be tiled side by side or one above the other, and dragged by their tab to re-dock. The layout is saved with the project.



An arrangement of three tiled trend graphs of a case-packing line — sealer, glue system and changeover signals: A and B measurement cursors with their measurement table on the first graph, stacked strips of digital and analog signals on the second, and two traces with their own axes on the third

Everything about trend graphs — strips, axes, cursors, notes, per-channel analysis and export — is described in the *Trends* chapter.

The menus

The Desktop has seven menus. Every entry is listed here with a one-line description and the chapter that describes it.

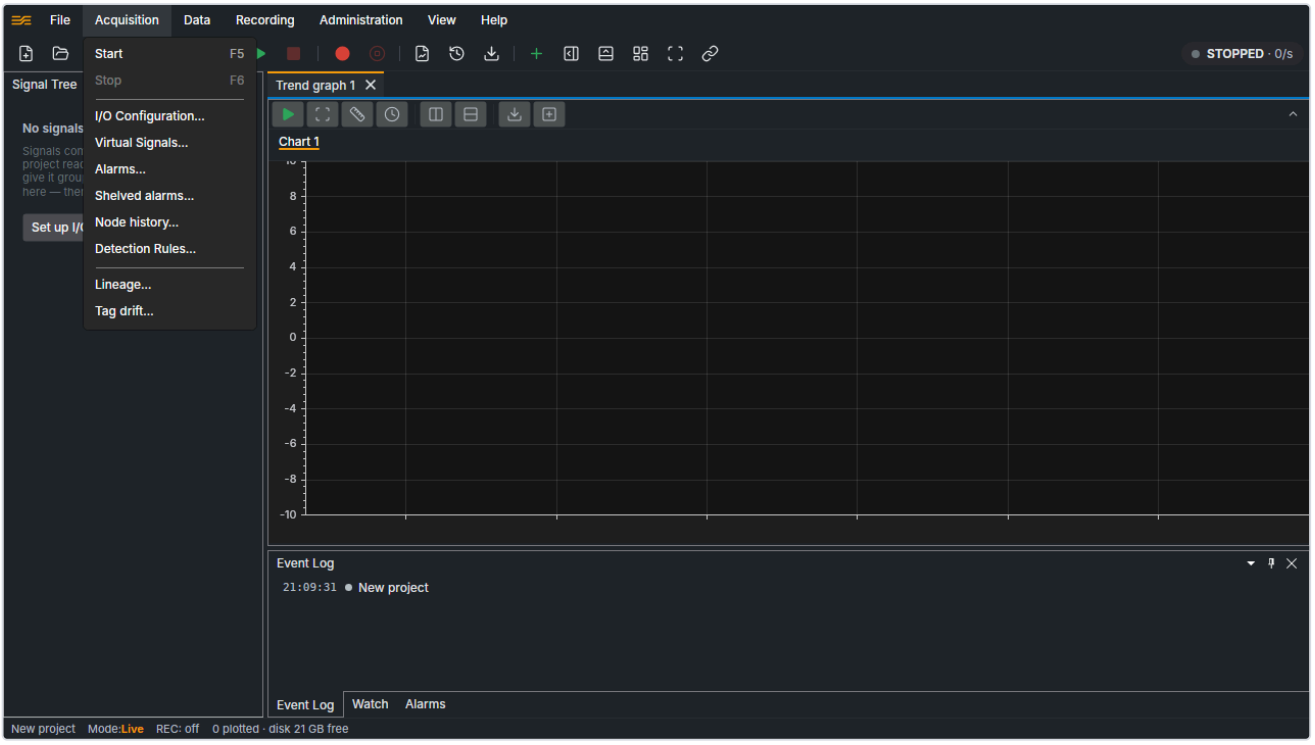
File

Entry	Shortcut	What it does
New Project	`Ctrl+N`	Starts an empty project
Open Project…	`Ctrl+O`	Opens a `*.meelmoproj` file
Save Project	`Ctrl+S`	Saves the open project
Save Project As…		Saves it under a new name
Open Recording…		Opens a `*.opslog` recording for playback (*Recording and playback*)
Close Recording		Leaves playback and returns to live. Enabled only in playback
Apply Analysis (chart preset)…		Applies a saved chart layout over the open project (*Trends*)
Save Analysis (chart preset)…		Saves the current chart layout on its own
Export Plotted Signals to CSV…	`Ctrl+E`	Exports the visible window of what is plotted

Entry	Shortcut	What it does
Export Plotted Signals to Excel…		The same, as an `.xlsx` workbook
Export Report (HTML)…		The charts and their statistics as an HTML document
Export Timing Evidence (CSV)…		Per-group timing quality of a recording. Enabled only in playback
Exit		Closes the application

Acquisition

The live pipeline: running it, and configuring what it reads and raises.



The Acquisition menu open over an empty project: Start and Stop; I/O Configuration, Virtual Signals, Alarms, Shelved alarms, Node history and Detection Rules; then Lineage and Tag drift

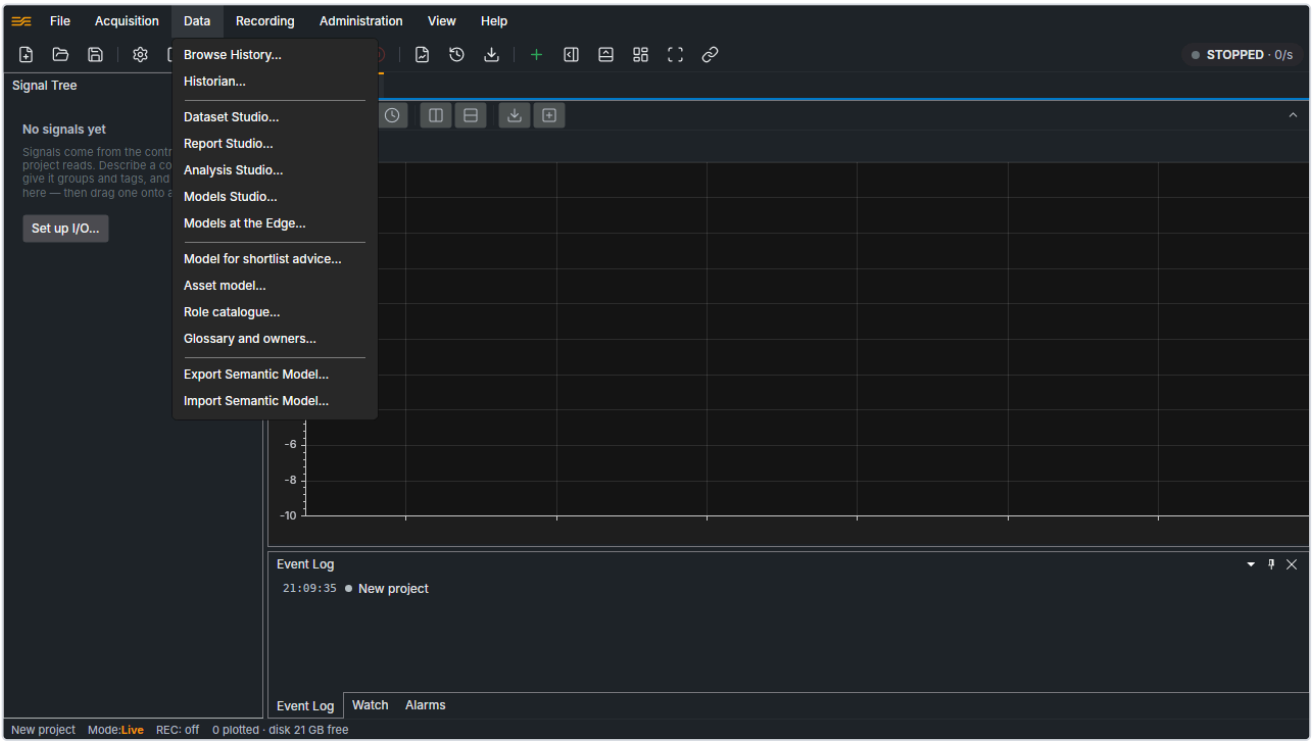
The menu has three groups, separated by lines: running acquisition, configuring what it reads and raises, and checking where signals come from and whether the controllers still match.

Entry	Shortcut	What it does
Start	F5	Starts acquisition. Disabled while acquiring
Stop	F6	Stops acquisition. Disabled while stopped
I/O Configuration…		Connections, acquisition groups, tags and circular buffers (*Acquisition and I/O configuration*)
Virtual Signals…		Computed signals (*Signals*)
Alarms…		Alarm rules and notifications (*Alarms*)
Shelved alarms…		What is shelved at the node, by whom and until when, and shelving an alarm (*Alarms*)
Node history…		The node's configuration revisions (*Configuration and deployment*)

Entry	Shortcut	What it does
Detection Rules…		Rules that turn signal behaviour into named executions, events and states (*The semantic model*)
Lineage…		Where a signal comes from and everything in Meelmo that reads it, worked out from the configuration the node is running (*Acquisition and I/O configuration*)
Tag drift…		Whether each device's tag table still matches the baseline its connections were written against (*Acquisition and I/O configuration*)

Data

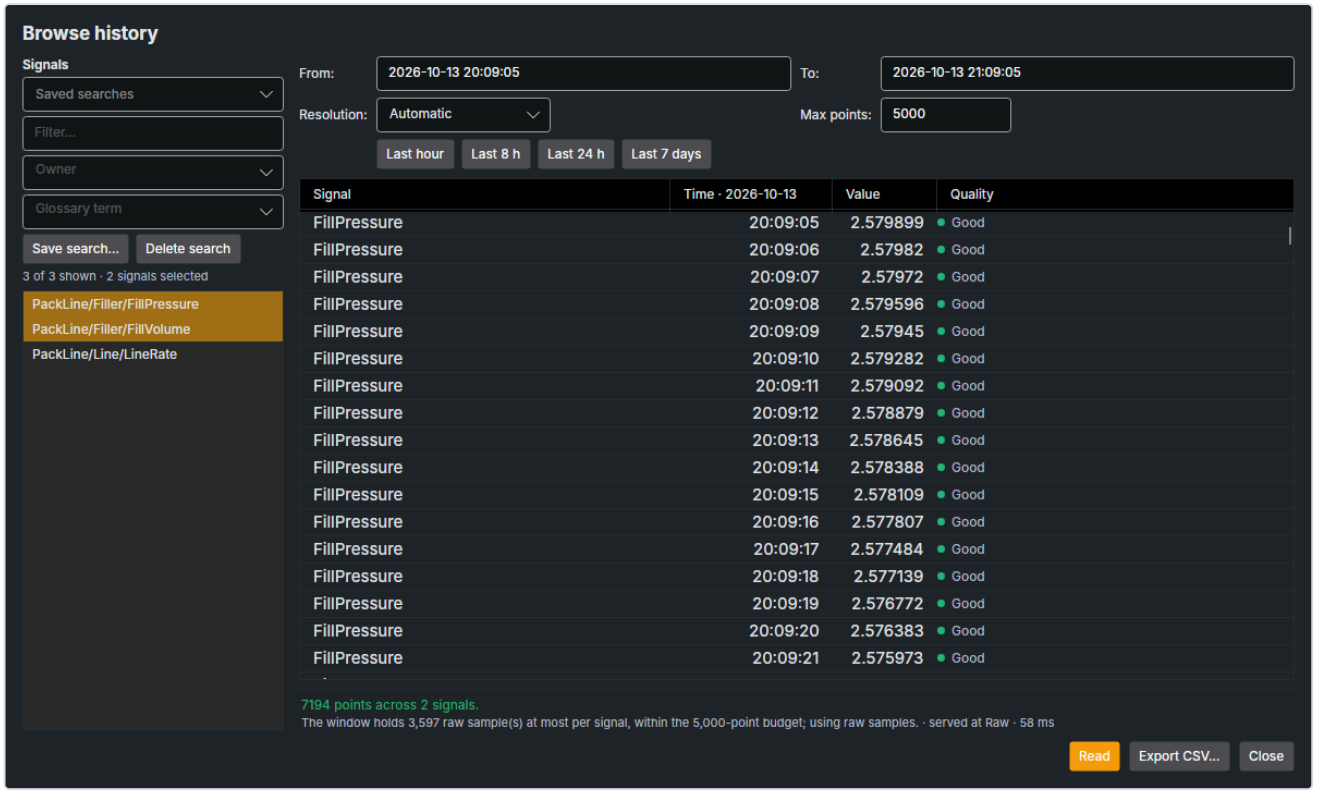
Where acquired data is stored and read, and what it means.



The Data menu open over an empty project: Browse History and Historian; the Dataset, Report, Analysis and Models Studios and Models at the Edge; Model for shortlist advice, Asset model, Role catalogue and Glossary and owners; then Export and Import Semantic Model

Entry	What it does
Browse History…	Queries the historian and plots the result (*The historian*)
Historian…	Chooses and configures the historian store, retention and rollups (*The historian*)
Dataset Studio…	Saved queries over history (*Datasets*)
Report Studio…	Report definitions (*Reports*)
Analysis Studio…	What changed around an event, and when (*Analysis*)
Models Studio…	Fitted models over history. Its functions need the Models module (*Models*)
Models at the Edge…	The inference runtime on the node, the models it holds and the versions running as model signals (*Models*)
Model for shortlist advice…	Which language-model provider, if any, the Models Studio's shortlist advice may be shown to

Entry	What it does
Asset model…	The plant's assets and the signals bound to them (*The semantic model*)
Role catalogue…	The kinds of asset and the roles their signals play (*The semantic model*)
Glossary and owners…	The site's terms with their synonyms and linked signals, and who answers for each signal (*The semantic model*)
Export Semantic Model…	Writes the semantic model to a file
Import Semantic Model…	Reads one back



Data ▸ Browse History with two signals of the filling line selected: From and To, Resolution and Max points, the Last hour to Last 7 days buttons, the samples with their quality, and a line saying how many points were read and at which resolution

Browse History…, shown above, is the quickest way to see what the historian holds for a signal; *The historian* chapter describes every field.

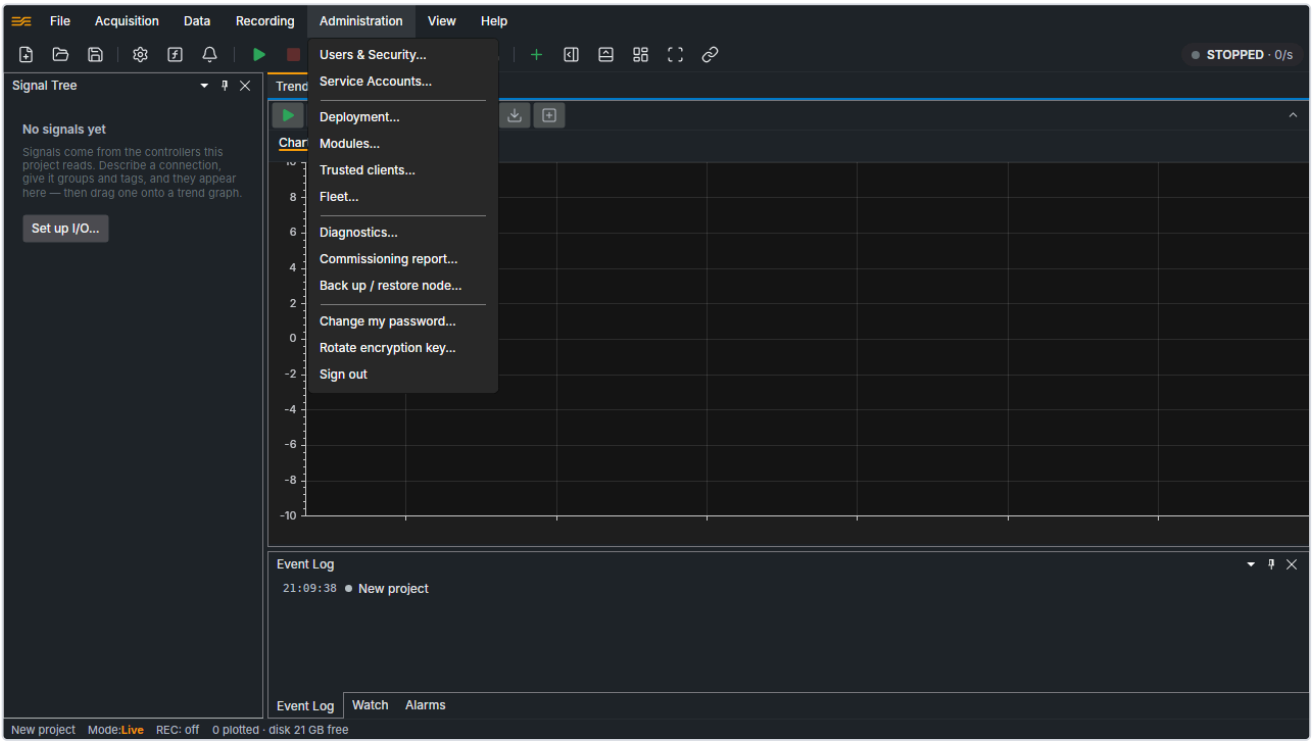
Recording

Entry	Shortcut	What it does
Start Recording	`Ctrl+R`	Starts writing a `*.opslog` recording
Stop Recording	`Ctrl+Shift+R`	Stops it
Continuous (rotating, record everything)		A toggle: record unattended into rotating, retention-managed files
Lossless (never drop samples, back-pressure)		A toggle: never drop a sample, at the cost of briefly holding acquisition

Entry	Shortcut	What it does
Recording settings…		Rotation size and age, retention and compression
Trigger…		Start and stop recording on a condition, with a pre-trigger buffer
Browse Recordings…		Recordings on disk, with span and size
Recorder Service…		Install, configure, start and stop the unattended recorder service
Recorder Status…		How the recorder service is doing
Recording Timing Analysis…		What a finished recording's timestamps are worth

All of these are described in the *Recording and playback* chapter.

Administration



The Administration menu, grouped into people, this node, health and evidence, and the signed-in session

Entry	What it does	Chapter
Users & Security…	Accounts, roles, the audit log	*Users and security*
Service Accounts…	Credentials for machine-to-machine access	*Users and security*
Deployment…	Standalone, server or client; where the node lives; certificates	*Configuration and deployment*
Modules…	Install, enable and disable optional modules	*Licensing and modules*
Trusted clients…	Which client certificates the node trusts	*Users and security*
Fleet…	Other nodes on the site	*Configuration and deployment*
Diagnostics…	Acquisition timing per connection and group	*Acquisition and I/O configuration*, *Diagnostics and support*

Entry	What it does	Chapter
Commissioning report…	The evidence document for handing a site over	*Diagnostics and support*
Back up / restore node…	A backup of the node's state, and restoring one	*Configuration and deployment*
Change my password…	Changes your own password	*Users and security*
Rotate encryption key…	Replaces the key protecting stored secrets	*Users and security*
Sign out	Returns to the sign-in window	this chapter

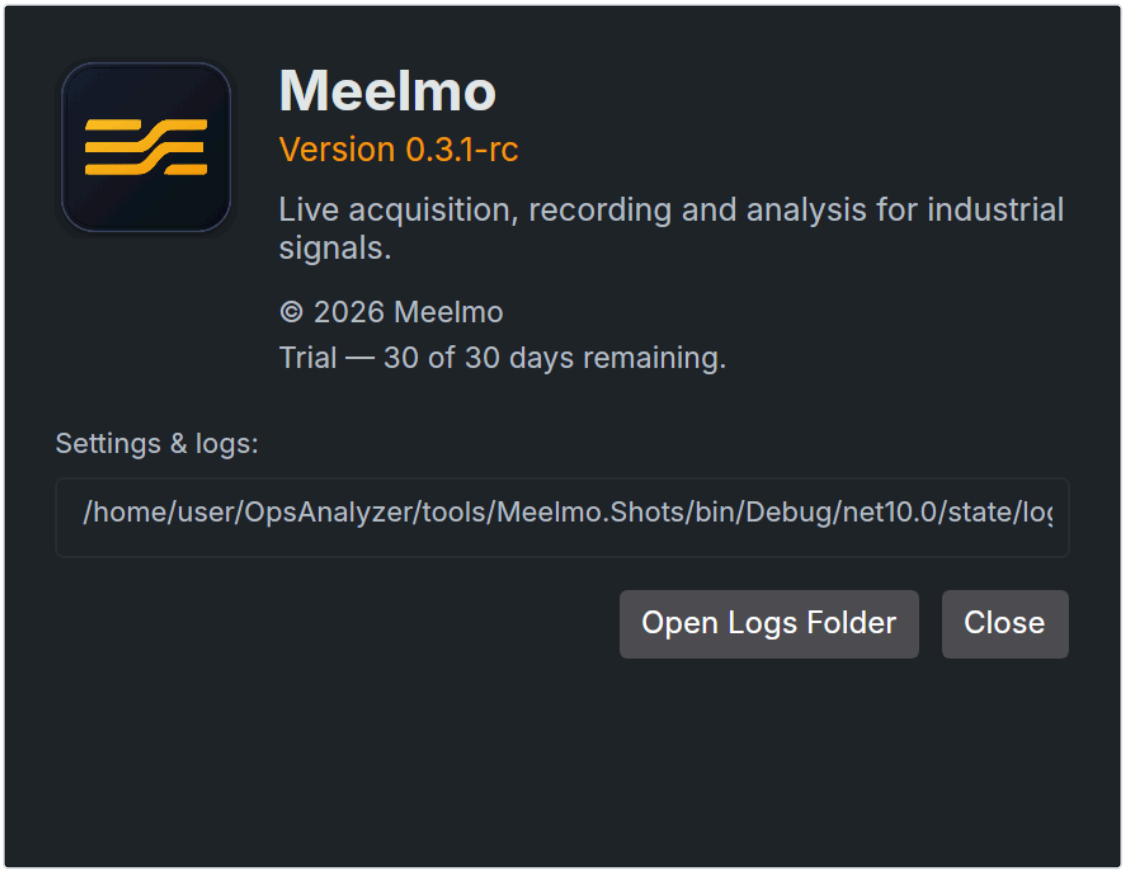
View

Entry	Shortcut	What it does
Add Trend Graph		Adds a new trend graph as a tab
New trend graph — split right		Adds one tiled to the right
New trend graph — split down		Adds one tiled below
Zoom to Fit		Fits every graph to its data
Clear Charts		Removes every trace from every graph
Panels ▸ Merge into one strip / **Stacked (shared time)** / **Grid**		Presents the plotted set in one strip, one pane per channel over a shared time axis, or a separate grid window
Sync Time Axis		A toggle: every graph shows the same time window
Scroll live charts continuously		A toggle: the live window follows the clock smoothly between samples, at the cost of redrawing 25 times a second
Draw live traces from kept columns		A toggle: draws analog traces from a decimation kept as readings arrive, which is much cheaper on a busy live chart
Signal Tree, **Event Log**, **Watch Window**, **Alarms Panel**		Show or hide each panel
Toolbar		Shows or hides the main toolbar. Remembered between sessions
Chart toolbar	`Ctrl+Shift+T`	Shows or hides the toolbar of every trend graph. Remembered between sessions
Full screen	`F11`	Full screen on or off. `Esc` always leaves full screen
Language		Switches between the ten languages the product ships in, without a restart (see *Language* below)
Toggle Light/Dark Theme		Switches the colour theme

Help

Entry	Shortcut	What it does
User Manual	`F1`	Opens the user manual in the built-in viewer

Entry	Shortcut	What it does
Demonstration…		Opens one of the simulated plants
Open Logs Folder		Opens the folder holding the application's log files
Save Support Bundle…		Saves a `.zip` of logs and configuration to send to support
Check for Updates…		Asks the configured update feed whether a newer release exists. The Desktop also checks quietly at start and notes a newer release in the Event Log
License…		What this installation is permitted, its Machine ID, and installing a licence (*Licensing and modules*)
About Meelmo…		Version, build and component information

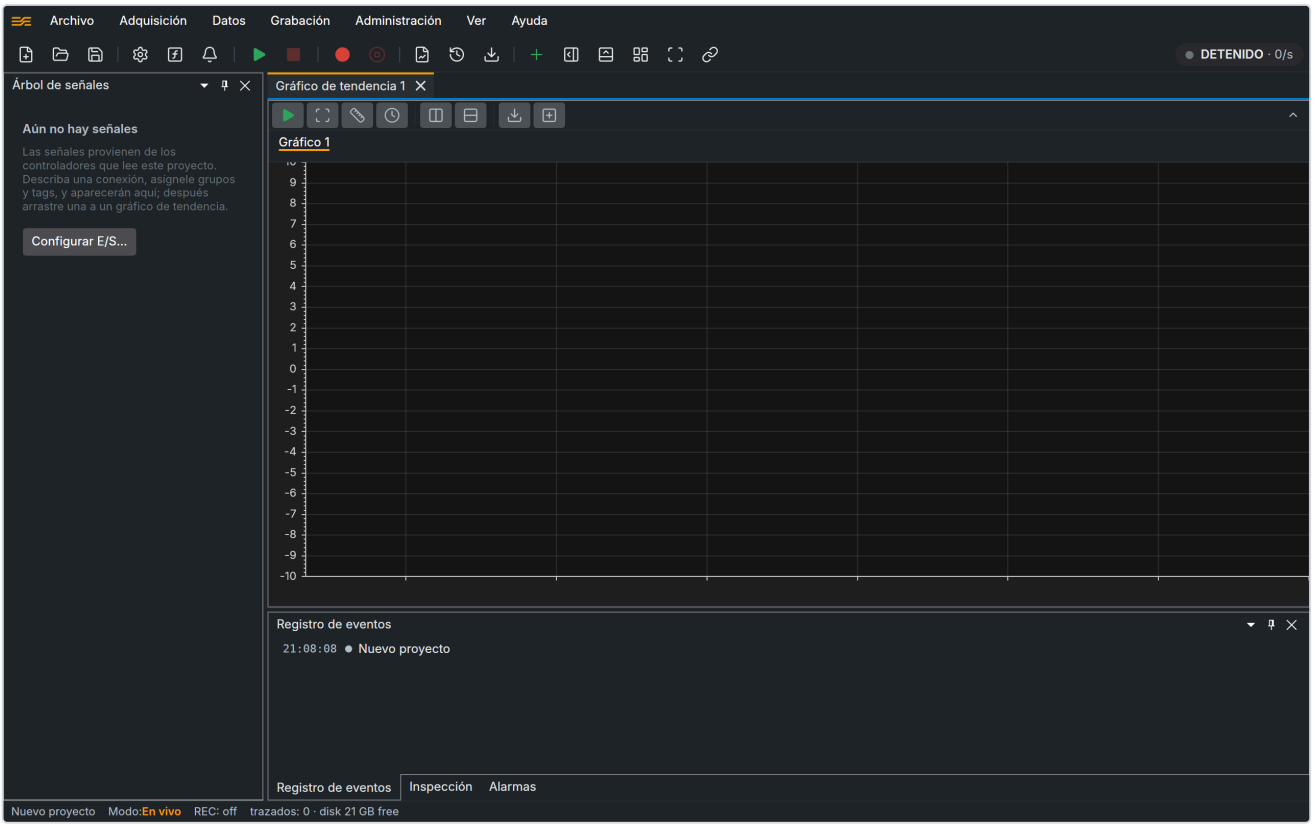


About Meelmo: the version, a one-line description, the licence state (here a trial with 30 of 30 days remaining), the folder holding settings and logs, and the Open Logs Folder and Close buttons

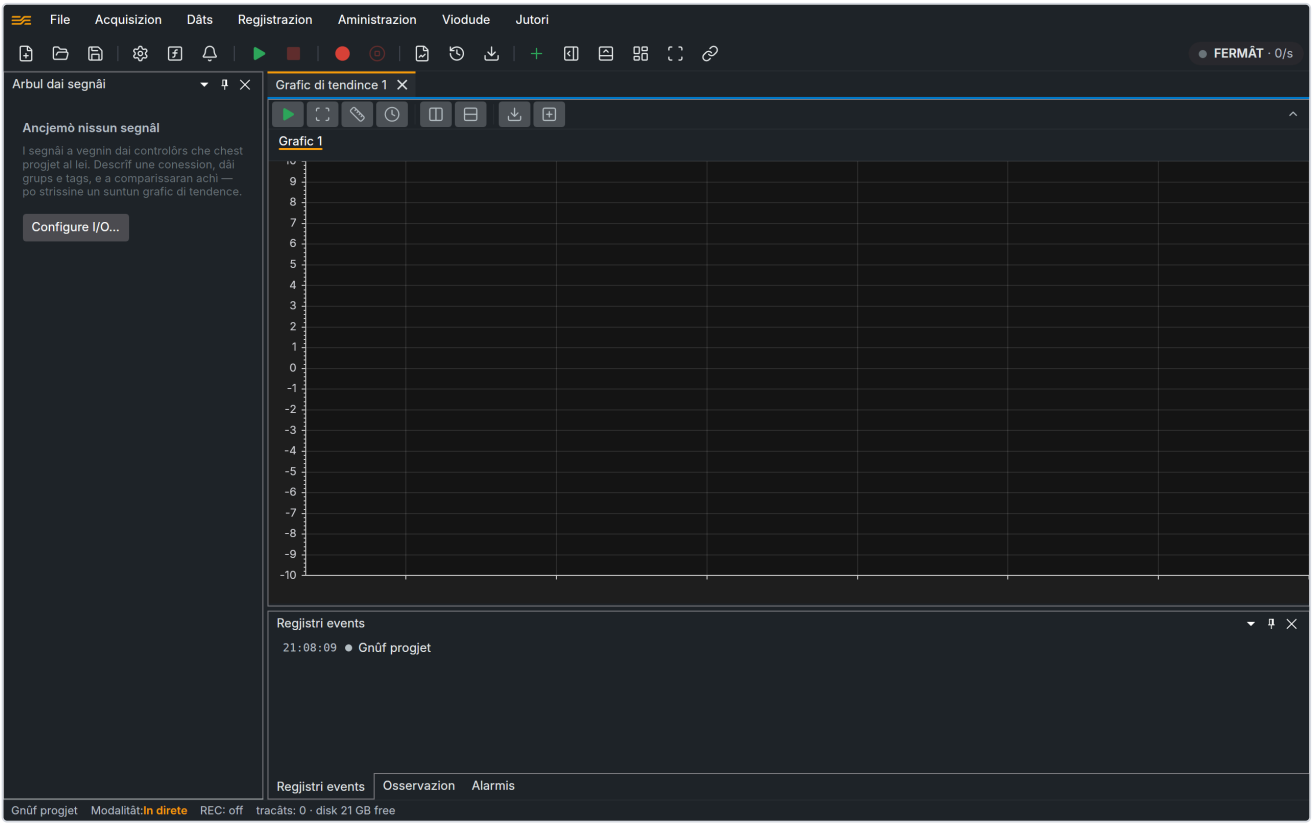
About Meelmo… is the quickest way to find the exact version for a support request and the folder where this installation keeps its settings and logs.

Language

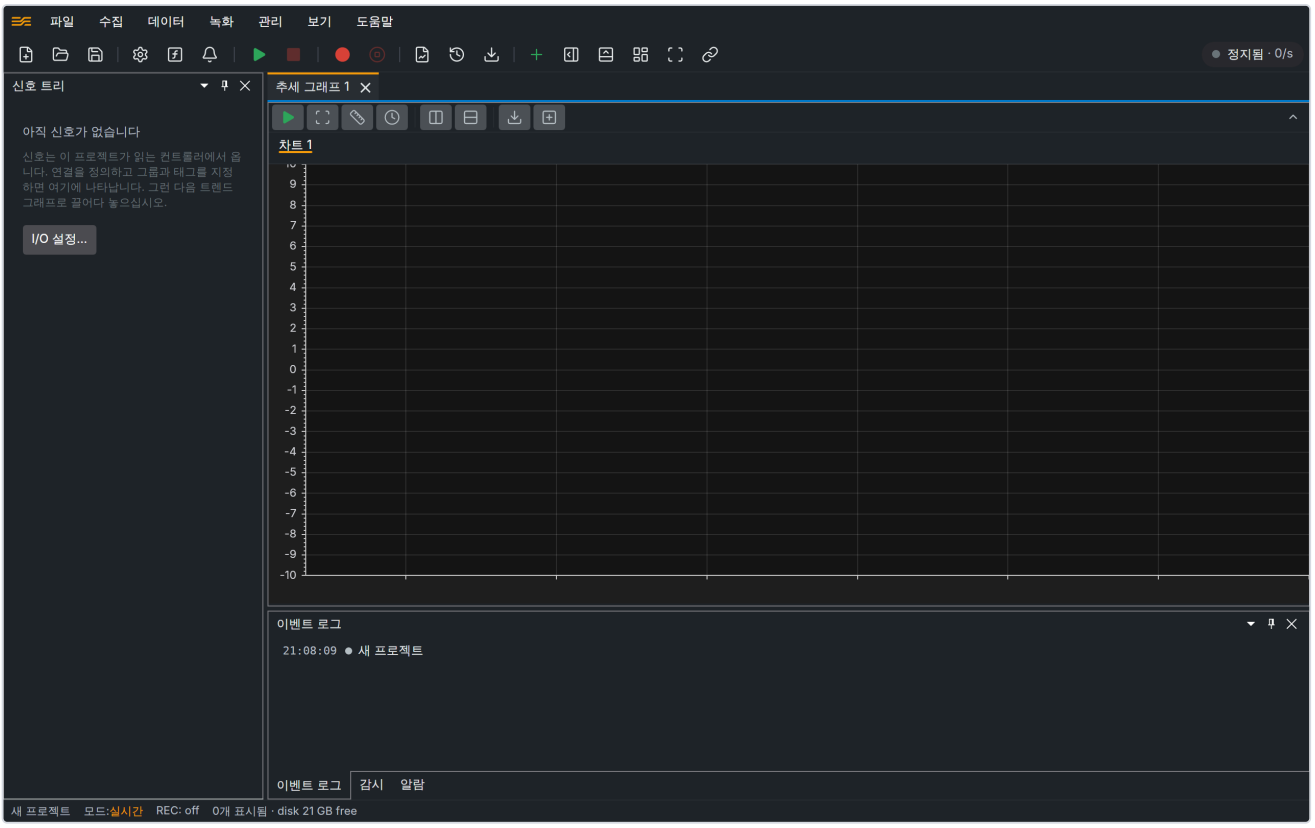
View ▸ Language lists the ten languages by their own names: English, Deutsch, Español, Français, Furlan, Italiano, 日本語, 한국어, Русский and 中文. The current one is ticked. Choosing another relabels the main window at once — menus, panel captions, the acquisition pill (**STOPPED** becomes *DETENIDO* in Spanish) and the status bar — without a restart, and the Desktop remembers the choice for the next start. Windows you open afterwards open in the new language, and messages written from then on are in it; lines already in the Event Log keep the language they were written in. The choice is stored with this installation's settings, so two people on two workstations attached to the same node can each work in their own language.



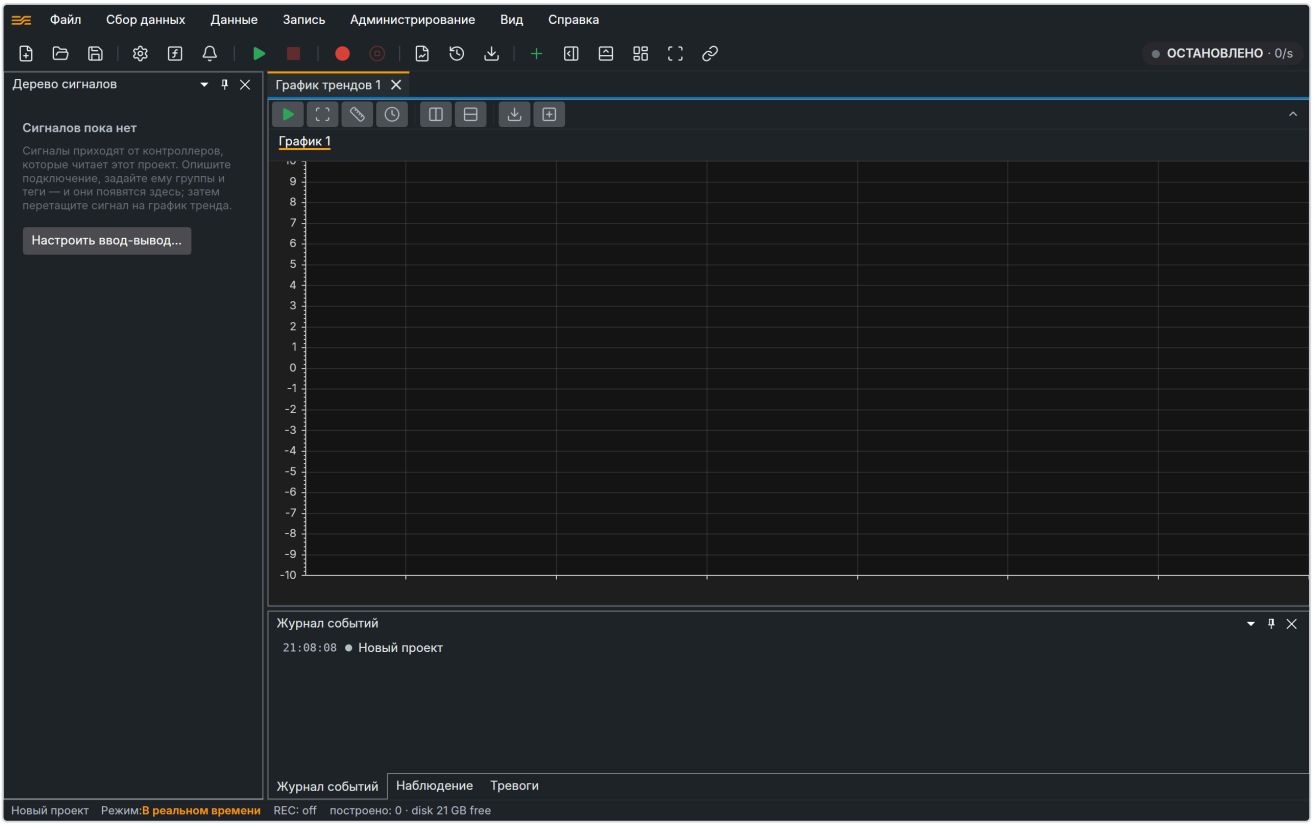
The empty Desktop in Spanish: Archivo, Adquisición, Datos, Grabación, Administración, Ver and Ayuda on the menu bar, Árbol de señales with Aún no hay señales and Configurar E/S..., Gráfico de tendencia 1 and Registro de eventos, and the DETENIDO pill



The same window in Furlan: Acuisizion, Dâts, Regjistrazion, Aministrazion, Viodude and Jutori, Arbul dai segnâi and the FERMÂT pill



The same window in Korean

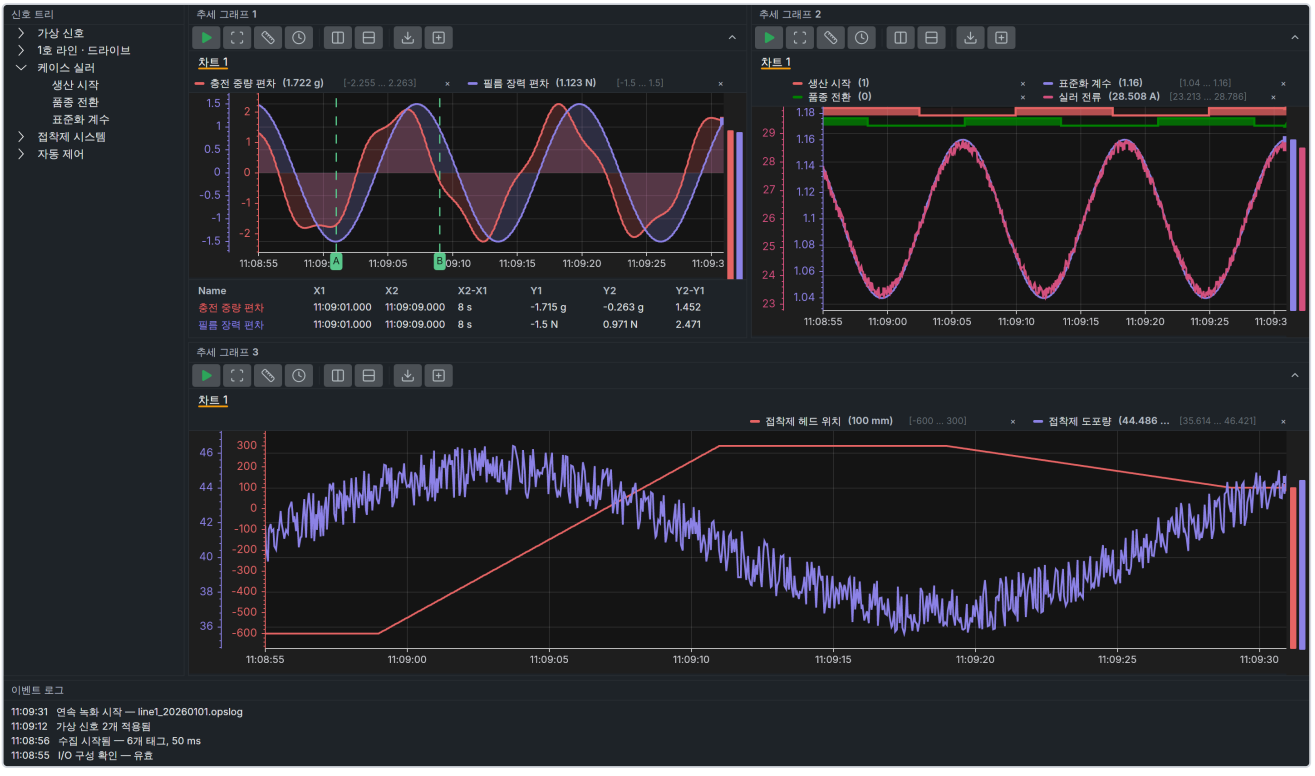


The same window in Russian

A full working layout reads the same way in every language. The next two figures show three tiled trend graphs, A and B measurement cursors with their table on the first, and the Event Log, once in Spanish and once in Korean:



Three trend graphs of a case-packing line in Spanish: Gráfico de tendencia 1 to 3, the A and B cursors eight seconds apart with the X1, X2, X2-X1, Y1, Y2 and Y2-Y1 table, and the Registro de eventos below



The same layout in Korean

View ▸ Toggle Light/Dark Theme switches between the dark and light colour themes in the same way.

The status bar

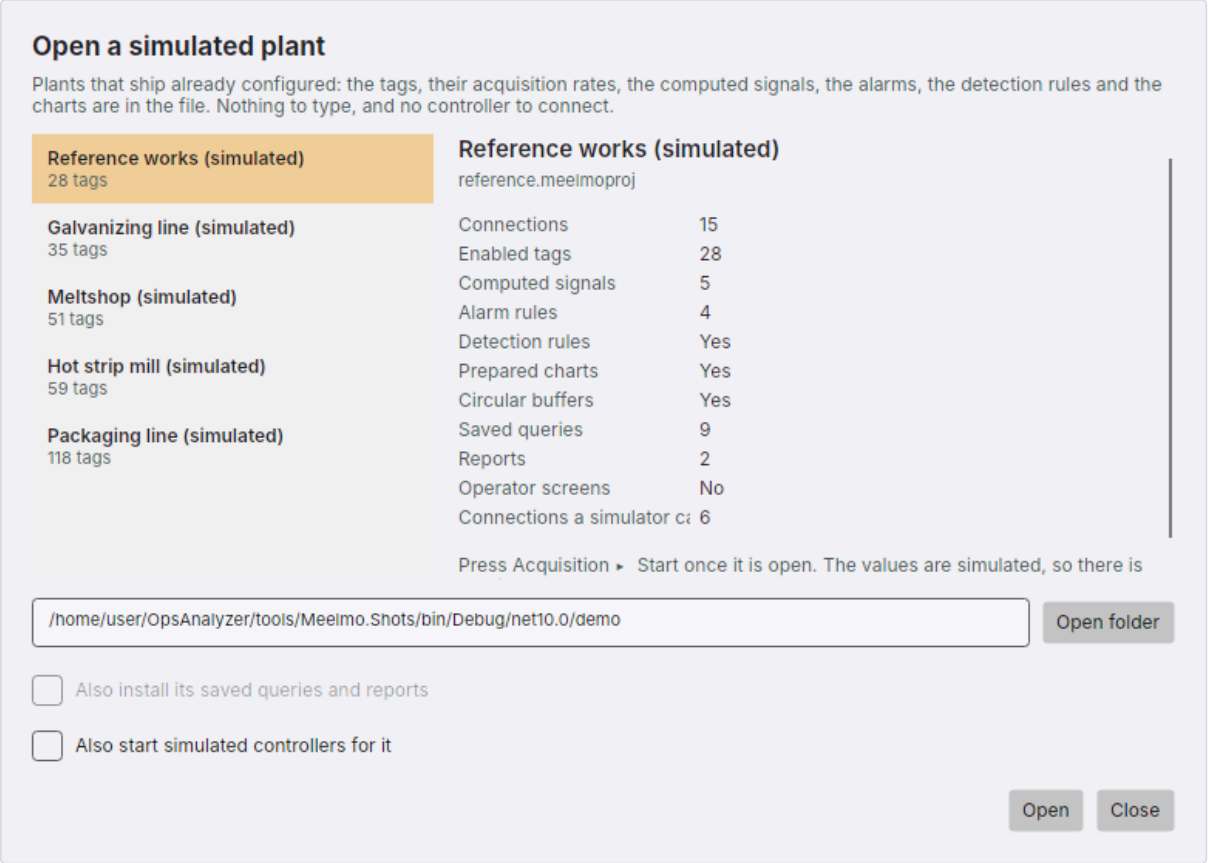
The status bar along the bottom of the window reads, left to right:

Item	Example	Meaning
Status message	`Configuration loaded from node 'Web Node'`	The last thing the application did
Mode	`Mode:Live`	`Live` when showing the plant, `Playback` when a recording is open
Recording	`REC: off`	The file being recorded to, or `off`
Health	`0 plotted · disk 20 GB free`	How many signals are on charts, dropped live batches if any, and free space on the recordings drive
Alarms		The number of active alarms and the highest severity, in red. Shown only while an alarm is active
Historian	`Historian: unavailable — …`	The historian's state. Shown only when the installation keeps a historian; drawn in red with the reason when there is a problem
Connection		On a client, the server it is attached to; in red when it is not attached. Not shown on a standalone installation
Load	`42 req/s · 3.1 KB/s · 1.8 ms read`	Driver requests per second, data rate and mean read time, while acquiring

In the figures of this chapter the historian item reads *Historian: unavailable — This workstation has no credential for the node's data API, so it can show no history. Open Deployment and save the connection again.…*. That is a client workstation that has not yet been given a data-access credential. The repair is to open **Administration ▸ Deployment…**, check the server account's password, and press **Save**; *Configuration and deployment*, under *When a client shows no history*, explains it step by step.

Opening a demonstration

The quickest way to see Meelmo working is to open one of the simulated plants.



The Open a simulated plant window with Reference works selected: its facts list 11 connections, 28 enabled tags, 5 computed signals, 4 alarm rules, detection rules, prepared charts, circular buffers, 9 saved queries and 2 reports

1. Choose **Help ▸ Demonstration…**.
2. Select a plant on the left. The list shows each plant's number of enabled tags. The right-hand side shows its file name and facts: **Connections**, **Enabled tags**, **Computed signals**, **Alarm rules**, **Detection rules**, **Prepared charts**, **Circular buffers**, **Saved queries**, **Reports**, **Operator screens** and **Connections a simulator can serve**.
3. Choose the options:
   - **Also install its saved queries and reports.** Datasets and reports are the node's configuration, not part of the project, so they cannot travel in the file. Ticking this installs them on the node as a recorded activation. Anything already saved under the same name is left alone. The option is disabled on a client workstation, because its saved queries and reports live on the node it is a client of and are installed from there.
   - **Also start simulated controllers for it.** Raises a Modbus weigher, a controller pushing cyclic telegrams, an OPC UA server, an IEC 60870-5-104 outstation, a DNP3 outstation and a BACnet/IP building controller on this machine, listening on `127.0.0.1` only, and switches on the connections that name them. Only the reference works has such connections. These are simulations of six protocols, not tested hardware.
4. Press **Open**. The project opens exactly as **File ▸ Open Project…** would open it.
5. Press **Acquisition ▸ Start** (`F5`). The values are simulated, so there is nothing to connect.

The path field shows the folder the plants were found in, and **Open folder** opens it. If the licence allows fewer tags than a plant enables, the window says so: the plant still opens and its configuration can be read, but **Start** refuses it.

Worked example: the packaging line

1. **Help ▸ Demonstration…**, select **Packaging line (simulated)** (120 tags), tick **Also install its saved queries and reports**, press **Open**.

2. Press `F5`. The acquisition pill turns to **ACQUIRING** and the rate climbs to a few hundred samples per second. The Event Log records the acquisition starting and the queries and reports installed (`Saved queries: N installed, 0 already there. Reports: …`).
3. The project opens with its prepared chart layout. In the Signal Tree, expand **Packaging ▸ Checkweigher** and double-click `CaseWeight` and `WeightTarget` to plot them on the active trend graph. The case weight moves around its target of `1240 g`.
4. Scroll the tree to **Virtual ▸ Computed** and double-click `WeightDeviation` — the difference between the two, in grams — to plot it too.
5. Open the **Alarms** tab under the graph. When `WeightDeviation` goes beyond ±15 g, the *Case weight out of tolerance* alarm appears there.

The project is now an ordinary project you can edit and, with **File ▸ Save Project As…**, save as your own.

Projects

A **project** (`*.meelmoproj`) describes a plant: its connections, acquisition groups, tags and circular buffers, its virtual signals, its alarm rules, its detection rules, its recording preferences and its saved chart layout.

Action	How
Start a new, empty project	**File ▸ New Project** (`Ctrl+N`) or the toolbar
Open a project	**File ▸ Open Project…** (`Ctrl+O`) or the toolbar
Save	**File ▸ Save Project** (`Ctrl+S`) or the toolbar. Needs the Engineer role
Save under a new name	**File ▸ Save Project As…**

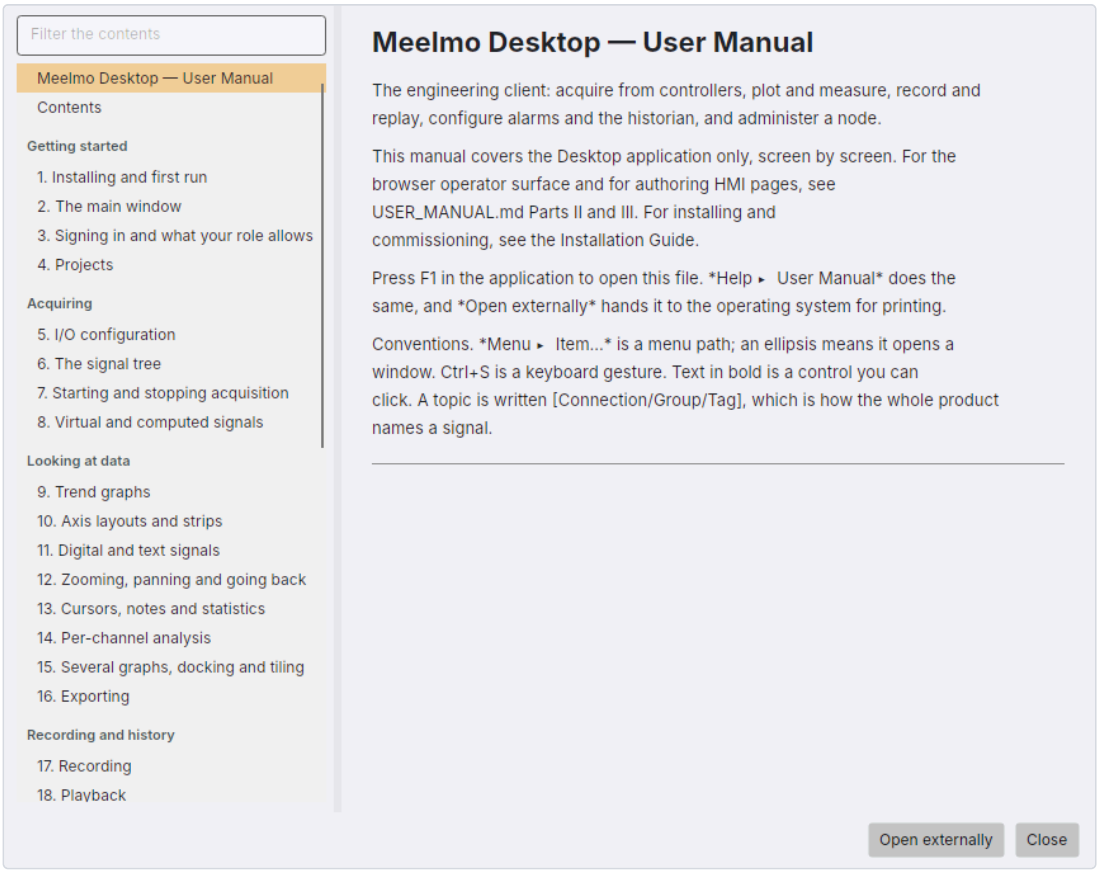
Points to know:

- **Where acquisition runs is not in the project.** Whether this machine acquires itself, serves others or is a client of a server is a property of the machine, set in **Administration ▸ Deployment**. The same project therefore opens standalone on a laptop and against a plant's server without being edited.
- **On a client, the project is the node's.** When the Desktop is attached to a node, the configuration shown is the node's (the status bar reads `Configuration loaded from node '…'`), and applying a change activates a new revision on the node, recorded in its history.
- **Where files are kept.** On Windows, Open and Save start in `C:\Meelmo\Data\projects\` unless a project is already open somewhere else. `C:\Meelmo\Data\` holds everything a commissioned site owns — licence, accounts, audit log, historian configuration — and is kept across upgrades and uninstalls.
- **The chart layout is saved with the project**: which signals are on which strip of which graph, their axes and notes. To keep a layout independently of a project, use **File ▸ Save Analysis (chart preset)…** and **Apply Analysis (chart preset)…**.

File	What it is
`*.meelmoproj`	A project
`*.opslog`	A recording
`*.meelmohmi`	A bundle of HMI pages, authored in the browser's HMI Studio

The built-in user manual

Help ▸ User Manual or `F1` opens the user manual in a viewer inside the application.



The built-in user manual viewer: a filterable table of contents on the left, the manual text on the right, and the Open externally and Close buttons

- **Filter the contents** narrows the table of contents on the left to the headings containing what you type.
- Click a heading to go to it.
- **Open externally** hands the manual to the operating system, for example to print it.
- **Close** closes the viewer.

The viewer shows the manual installed beside the application. If none is found, the Desktop says so. A manual delivered as a PDF opens in the operating system's PDF viewer instead.

Keyboard shortcuts

Gesture	Does
`Ctrl+N`	New project
`Ctrl+O`	Open project
`Ctrl+S`	Save project
`Ctrl+E`	Export plotted signals to CSV
`Ctrl+R`	Start recording
`Ctrl+Shift+R`	Stop recording
`Ctrl+Shift+T`	Show or hide the chart toolbars
`F1`	User manual
`F5`	Start acquisition

Gesture	Does
`F6`	Stop acquisition
`F11`	Full screen on or off
`Esc`	Leave full screen

Within a chart: drag to pan, scroll the mouse wheel to zoom, right-click for the chart's context menu. In the Signal Tree: double-click a tag to plot it on the active graph, or double-click a plotted tag to remove it; drag a tag onto a graph to plot it there.

Acquisition and I/O configuration

Acquisition is how values get from a controller into Meelmo. This chapter describes how a plant is described to Meelmo — connections, drivers, acquisition groups, tags and circular buffers — how acquisition is started and stopped, and how to prove that it is sampling the way it was configured to.

Configuring I/O needs the **Engineer** or **Administrator** role. Starting and stopping acquisition needs at least the **Operator** role.

How acquisition is organised

A plant is described in three levels, with a fourth where a controller has one:

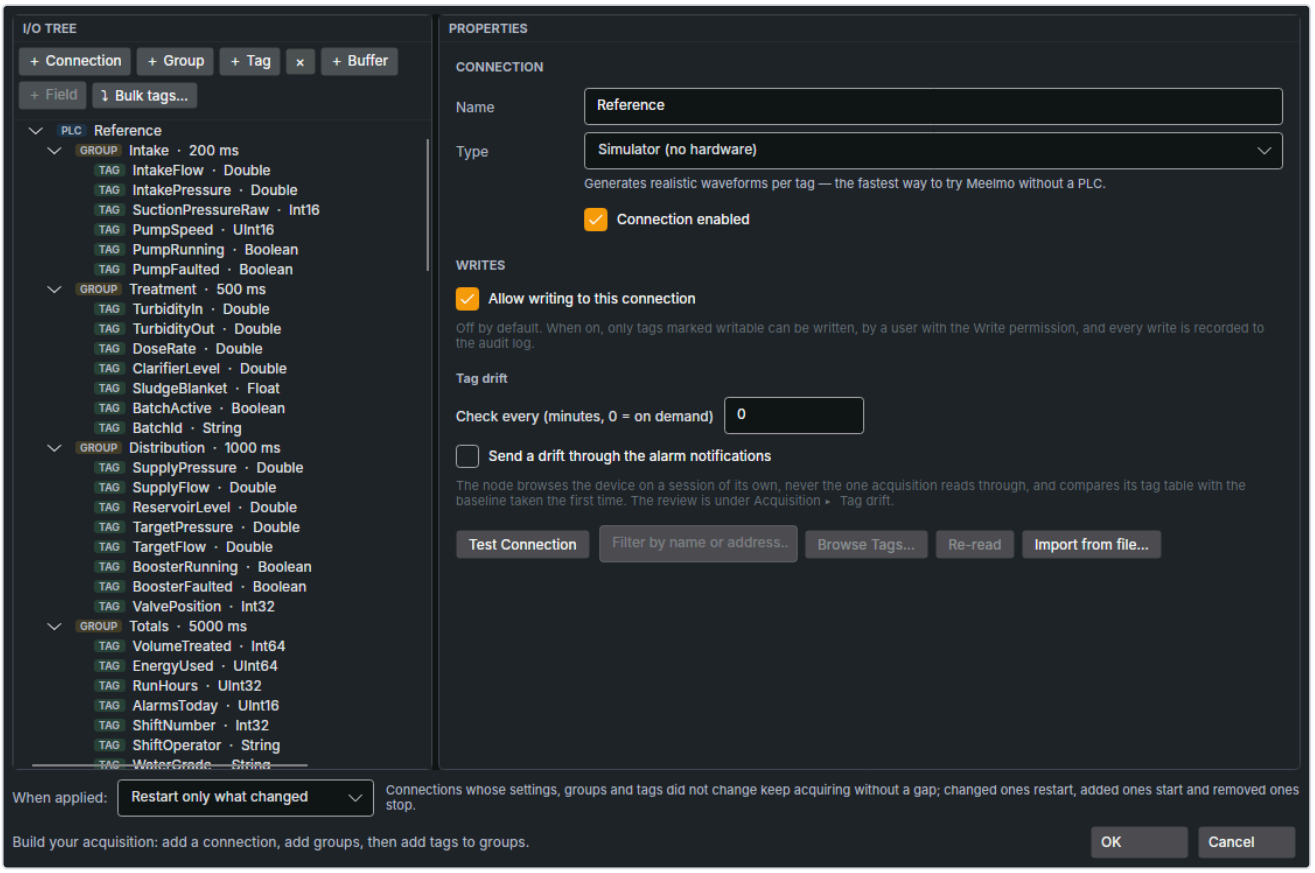
Level	What it is	Example from the packaging line
Connection	One controller or endpoint: a driver type, its address and settings	`Packaging`, using the Simulator driver
Acquisition group	A set of tags read together in one driver call, at one poll rate, sharing one timestamp	`Checkweigher`, read every 500 ms
Tag	One signal: a name, an address, a data type, a unit	`CaseWeight`, a `Double` in `g`
Circular buffer	A ring the controller fills faster than anything can poll it, drained into ordinary tags	none on the packaging line; three on the reference works

Every tag is known throughout the product by its topic, `Connection/Group/Tag` — `Packaging/Checkweigher/CaseWeight`. That name is what trend graphs, alarm rules, datasets, reports, the historian and operator screens refer to. **Renaming a connection, group or tag therefore changes the signal's identity**; see *Removing and renaming* below.

The I/O Configuration window

Open it with **Acquisition ▸ I/O Configuration…**, the I/O configuration button on the toolbar, **Set up I/O…** in an empty Signal Tree, or right-click in the Signal Tree ▸ **Open I/O configuration…**. Right-clicking a connection or a tag and choosing **Edit connection…** or **Edit tag…** opens it with that item selected.

Small changes do not need the window at all. The Signal Tree's right-click menu adds connections and tags, switches a connection on or off, and deletes a connection or a tag. Each of these is validated, asked about and applied exactly as **OK** applies an edit here; see *Signals ▸ Configuring I/O from the tree*.



The I/O Configuration window for the packaging line: the I/O tree on the left shows the Packaging connection with its Erector, BottomSealer, Loader and Checkweigher groups at 500 ms and their tags with data types; the Properties pane on the right shows the connection's name, its type Simulator (no hardware), Connection enabled, and Allow writing to this connection

The window has two panes and a footer.

The I/O tree (left) shows every connection (**PLC**), acquisition group (**GROUP**, with its poll rate) and tag (**TAG**, with its data type) of the project. Select an item to edit it on the right. Its buttons:

Button	What it does
+ Connection	Adds a new connection
+ Group	Adds an acquisition group to the selected connection
+ Tag	Adds a tag to the selected group (or connection)
×	Removes the selected item(s)
+ Buffer	Adds a circular buffer to the selected connection
+ Field	Adds a field to the selected buffer's record. Enabled only when a buffer is selected
⤵ Bulk tags…	Adds many tags at once by pattern, pasted list, Excel workbook or script

Tags can be multi-selected in the tree (Ctrl-click, Shift-click) and right-clicked: **Move N tags to group**, **Edit N tags…**, **Group by folder…** and **Delete N tags** act on the selection. A branch of the tree draws its first 250 tags and then a row, *… 7,566 more tags, in the tag table*; a group of more than 50 tags starts closed. Choosing that row opens the **Tag table** on that group.

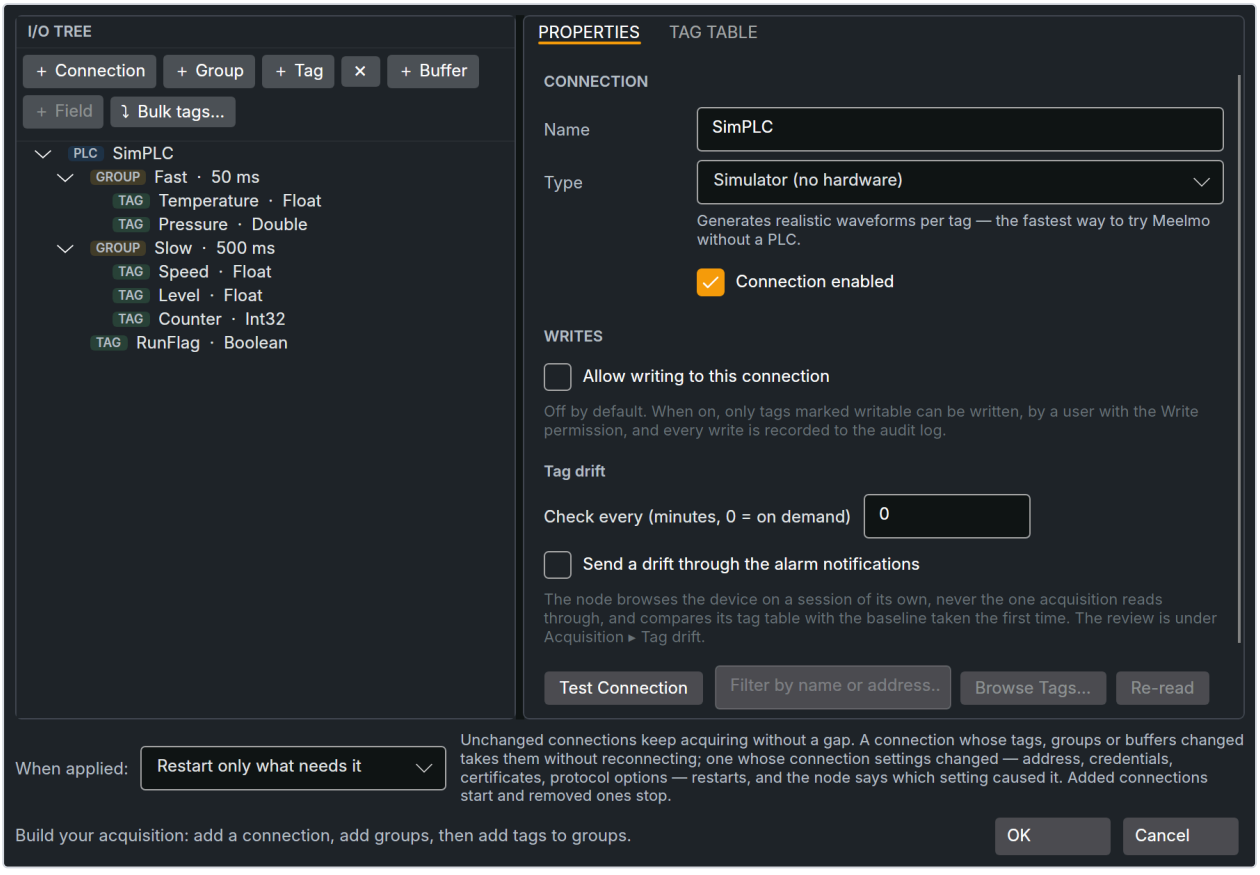
The right-hand pane has two tabs. **Properties** shows the fields of whatever is selected: a connection, a group, a tag, a buffer or a buffer field. Each is described below. **Tag table** lists the tags of the selected item's connection; see *Working with many tags*.

The footer reads *Build your acquisition: add a connection, add groups, then add tags to groups.* and carries **OK** and **Cancel**. Problems found while editing are listed under **Validation**.

OK validates the whole configuration and applies it. **Validation is not a formality**: OK refuses a configuration that cannot work — a duplicate connection name, a tag with no address, a type that contradicts its address — and says which item and which field. **Cancel** discards every change made since the window opened.

> **Applying an I/O configuration restarts a connection only when its session has to change.** Adding, removing or editing tags, groups or buffers on a running connection is taken while it stays connected, and the tags you did not change keep acquiring with no gap. Changing a connection's endpoint, credentials, certificates, security or protocol options restarts that connection, and the node says which setting caused it; plan those the way you would plan any other restart of the line's data collection. On a client attached to a node, OK activates a new configuration revision on the node, recorded with your name in **Acquisition ▸ Node history…**.

The figure below, from an earlier build and a small test project, shows the same window with a two-group connection `SimPLC` and one ungrouped tag.



The I/O Configuration window from an earlier build: the SimPLC connection with a Fast group at 50 ms (Temperature, Pressure), a Slow group at 500 ms (Speed, Level, Counter) and an ungrouped RunFlag tag

Connections

A connection is one controller or endpoint. Select it in the I/O tree to see its properties.

Field	Meaning
Name	The connection's name, and the first part of every topic it produces. Must be unique in the project
Type	The driver. The list shows every driver this installation carries, with a one-line description under the picker
Connection enabled	Cleared, the connection is kept in the project but not acquired. Used for a controller that is not yet on the network
CONNECTION PARAMETERS	The driver's own settings: address, port, rack and slot, security and so on. Each driver ships sensible defaults
Allow writing to this connection (under **WRITES**)	Off by default. When on, only tags marked writable can be written, only by a user with the permission to command a tag, and every write is recorded in the audit log

The buttons under the settings:

Button	What it does
Test Connection	Connects with the current settings and reads one tag (whichever is first). Proves the socket opens and the controller answers
Filter by name or address	Narrows the browse result to tags whose name, address or comment contains the text
Browse Tags…	Reads the controller's own symbol table and lets you pick tags from it. Only drivers that can browse enable it
Re-read	Walks the device again and replaces the cached tag list
Import from file…	Reads a tag list from an engineering export — a TIA Portal tag table, a STEP 7 symbol file, a Studio 5000 tags export or `.L5X` project, a plain list, or an OPC UA NodeSet — and previews it against this connection before adding anything, with no connection to the device (see *Importing a PLC tag list*)

The drivers

Every driver the product has, by the name the **Type** list shows:

Driver	Protocol	Acquisition mode	Browses tags	Notes
Simulator (no hardware)	—	Polling	—	Built-in waveform generator; no hardware needed. The demonstration plants use it
Siemens S7	S7 / RFC1006 (ISO-on-TCP)	Polling	—	S7-200/300/400/1200/1500, read and write
Siemens S7-1200/1500 (S7CommPlus, optimized)	S7CommPlus / RFC1006	Polling	yes	Optimized data blocks by symbol name; TLS 1.3 inside ISO-on-TCP. **Preview**: validation against physical controllers is still in progress
Modbus TCP	Modbus TCP, RTU over TCP	Polling	—	Coils, discrete inputs, holding and input registers

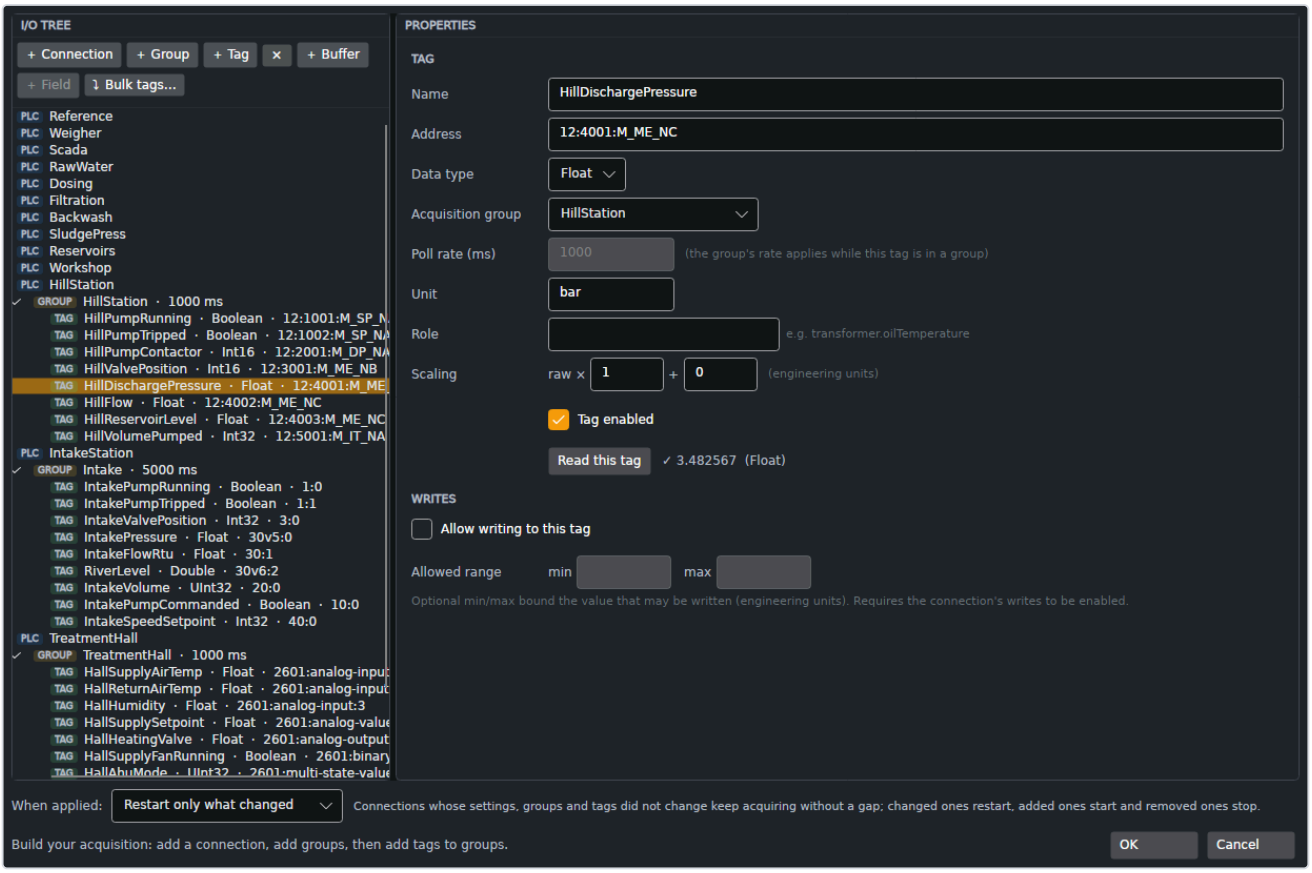
Driver	Protocol	Acquisition mode	Browses tags	Notes
Modbus RTU (serial)	Modbus RTU over RS-485 or RS-232	Polling	—	The same addresses, data types and grouped reads as Modbus TCP, through a COM port or a Linux tty. Connections on one port share it and take turns; `5:40001` addresses unit 5. Tested against a simulated slave and through a Linux pseudo-terminal pair, not yet a physical device
SunSpec device (Modbus)	SunSpec models over Modbus TCP, RTU over TCP	Polling	yes	Solar inverters, MPPT trackers, meters, DER and batteries. The map is found on connect (`SunS` at 40000, 50000 or 0) and its model chain walked; tags name SunSpec points — `inverter/W` , `mppt/1/DCW` , `meter/TotWhImp` — read with their scale factors applied, and a point the device does not implement is Bad saying so. Several devices behind one gateway are one connection with several unit ids. Writes only the active-power limit and `Conn` . Tested against a simulated device and maps encoded by pysunspec2, not yet a physical inverter
M-Bus meters (EN 13757)	M-Bus (EN 13757-2/-3) on a serial level converter, or a TCP gateway	Polling	yes	Heat, water, gas and electricity meters on a wired M-Bus. Meters by primary address (0–250) or secondary address, wildcards allowed; every telegram of a meter is read, and its records are named from what they hold — `energy` , `volume` , `flow_temperature` , `energy@storage1` — with their units. The meter's status byte sets the quality; telegrams encrypted with AES (security mode 5 or 7) are read with the meter's key, other encryption is reported, not read. Read-only. Tested against simulated meters and libmbus's readings of 78 real meters' telegrams, not yet a physical meter
Allen-Bradley (EtherNet/IP)	EtherNet/IP (CIP)	Polling	yes	ControlLogix, CompactLogix and Micro800 by tag; MicroLogix, SLC 500 and PLC-5 by data-table address (`N7:0` , `F8:10`). The controller family is a connection setting
Beckhoff TwinCAT (ADS)	ADS	Polling	yes	TwinCAT symbols
Mitsubishi MELSEC (MC)	MELSEC MC, binary over TCP	Polling	—	Bit and word devices
Omron FINS	FINS/TCP (with the node-address handshake) or FINS/UDP, port 9600	Polling	a span of addresses	Omron CS, CJ and CP PLCs, and NJ/NX through their FINS memory areas. Addresses as CX-Programmer writes them: `D100` , `D100.05` , `CIO 0.03` (or `0.03`), `W10` , `H20` , `A100` , `E0_100` (EM bank in hexadecimal), `T5` , `C12` . Tags in one area are read in contiguous blocks of up to 990 words. A refused block is read again tag by tag, so one address past the end of a smaller CPU is Bad on its own, with the FINS end code in words

Driver	Protocol	Acquisition mode	Browses tags	Notes
OPC UA server	OPC UA binary	Polling, Subscription	yes	Tags referenced by NodeId. The built-in client signs and encrypts with Basic256Sha256, Aes128_Sha256_RsaOaep or Aes256_Sha256_RsaPss and checks the server's certificate (see *OPC UA message security* below); on policy None it refuses cleartext credentials unless allowed. Polls in batches, or subscribes when **Subscriptions** is on (see *OPC UA subscriptions* below)
MQTT PLC	MQTT	Subscription	—	Tag values published to a broker; TLS and QoS; keeps message time and sequence
MTConnect agent	MTConnect 1.3–2.x over HTTP	Subscription	yes	Machine tools through their MTConnect agent. Read-only; follows the agent's sequence numbers so no change between requests is missed, and keeps the agent's timestamps. Tested against the standard's example documents and a simulated agent, not yet a physical machine
BACnet/IP	BACnet/IP (ASHRAE 135, Annex J) over UDP	Polling, Subscription	yes	Building controllers. Read-only: devices found with Who-Is, read with ReadPropertyMultiple or followed by change-of-value subscription; each object's status flags become the reading's quality. Tested against the standard, a simulated device and BACpypes3, not yet a physical controller; not BTL-tested. BACnet/SC is not supported
IEC 60870-5-104 outstation	IEC 60870-5-104 over TCP, optionally TLS (IEC 62351-3)	Subscription	yes	Substations and telemetry outstations. Read-only: a general interrogation on connect, then every change the outstation sends, with its own time tags; a second address can form a redundancy group. Tested against the standard, a simulated outstation and an independent IEC 104 stack (hat-drivers), not yet a physical RTU
DNP3 outstation	DNP3 (IEEE 1815) over TCP	Subscription	yes	RTUs and outstations in water and power. Read-only master: an integrity poll on connect, then class 1–3 events by polling and unsolicited response, with the outstation's own event times; reads everything again after a restart or an event-buffer overflow. No Secure Authentication. Tested against the standard, a simulated outstation and opendnp3, not yet a physical RTU
Tasmota device	Tasmota over MQTT, or its HTTP `/cm` endpoint	Subscription, Polling	yes	Relays, plugs, sensors and energy meters running the open-source Tasmota firmware. Over MQTT, telemetry arrives as the device publishes it, its Last Will makes every value Bad, and silence for 2.5 TelePeriods makes them Uncertain; over HTTP the device is polled. Writes only an output (`POWER1` – `POWER32`) or `Dimmer` . Tested against a simulated device on a real broker and a simulated web server, not yet a physical device

Driver	Protocol	Acquisition mode	Browses tags	Notes
Sparkplug B	Sparkplug B 3.0 over MQTT	Subscription	yes	A Sparkplug namespace other edge nodes publish, recorded as a host application. Births declare the metrics and their aliases; data by alias or name; the edge node's own metric timestamps are the source time. A sequence gap or an alias no birth declared makes that edge node's values Uncertain until its next birth; its death certificate, matched to the session by bdSeq, makes them Bad. Templates become their members, arrays their elements; DataSets, Bytes and Files are skipped with a notice. Listens only unless set otherwise. Tested against the node's own Sparkplug edge node and a hand-built one on a real broker, not yet a third-party edge node
Shelly device (Gen2+)	Shelly's JSON-RPC API over MQTT, or its HTTP `/rpc` endpoint	Subscription, Polling	yes	Shelly Plus, Pro, Gen3 and Gen4 relays, plugs, energy meters and sensors (not Gen1). Over MQTT, NotifyStatus arrives as the device sends it, stamped with the device's UTC `ts` while its clock is synchronised; its `online` topic going false makes every value Bad, and silence past the stale period — after the device has been asked for its status and not answered — makes them Uncertain. Over HTTP `Shelly.GetStatus` is polled, with SHA-256 digest authentication. Writes only `switch:<id>/output`. Tested against simulated devices on a real broker and a simulated web server, not yet a physical device
KNX/IP	KNXnet/IP: a tunnel over UDP or TCP to a KNX IP interface or router, or routing on the multicast group	Subscription, Polling	yes	Lighting, blinds, heating and metering on a KNX bus. Listens to the group telegrams the bus's devices send — a tag is a group address and a datapoint type, `1/2/3@9.001` — and sends a GroupValueRead only for tags marked `;read`, at most 20 telegrams a second. The tunnel is checked every 60 s; a lost link makes every value Bad with the reason, a tag never heard is Uncertain. Values are stamped on arrival. Writes only GroupValueWrite to a tag's group. KNX IP Secure for a TCP tunnel (the tunnel's user, password and device authentication code) and for routing (the backbone key), typed in or from an ETS keyring; not KNX Data Secure. Plain KNXnet/IP belongs on an isolated network. Tested against xknx's vectors, a simulated interface, knxd and xknx's secure routing, not yet a physical interface

Driver	Protocol	Acquisition mode	Browses tags	Notes
LoRaWAN (network server)	LoRaWAN, read through a network server's MQTT integration: ChirpStack v4 or The Things Stack v3	Subscription	yes	Battery sensors on a LoRaWAN network. Not a radio stack: one application's uplinks as the network server publishes them, decoded by the server's codec, with RSSI, SNR, the frame counter and the battery beside them and the server's UTC uplink time as the source time. An uplink delivered twice is read once; a frame counter that goes back is logged as a device reset; values turn Uncertain after 2.5 expected uplink intervals with nothing heard. Read-only: downlinks are never sent. Tested against both servers' documented payloads and a simulated network server on a real broker, not yet a real network server or device
IO-Link master (JSON)	An IO-Link master's JSON interface over HTTP or HTTPS: the IO-Link Community's JSON Integration, or ifm's IoT Core	Polling	yes	IO-Link sensors on a master's ports — not the IO-Link wire, which the master runs. Each port's status and device (vendor and device id, product, serial), its input process data as hex or decoded by a typed-in bit layout or the device's IODD, and ISDU parameters on request. A port without a device, a device other than the one expected, process data the master flags invalid, a short circuit in its event log and an unreachable master are Bad; a port whose read fails while the master answers is Uncertain for a while. Read-only. Tested against the specification's and ifm's documented payloads and a simulated master, not a physical one
Generic Cyclic UDP	OACU telegrams over UDP	Cyclic streaming	—	Passive receiver for controller-pushed telegrams (Siemens, Allen-Bradley, Beckhoff, CODESYS); 2–5 ms, 1 ms with a fast task
Generic Cyclic TCP	OACU telegrams over TCP	Cyclic streaming	—	The same frame over TCP; connects out or accepts one inbound connection, and reconnects
Custom TCP protocol	Pluggable TCP	Polling	—	A request/response extension point for a protocol you supply

Add-on drivers are discovered at start-up and appear in the same list. On the Community edition only the Simulator, Modbus TCP/RTU, OPC UA, MQTT, MTConnect, IEC 60870-5-104, DNP3, BACnet/IP, Tasmota, Sparkplug B, Shelly, SunSpec, LoRaWAN, IO-Link, M-Bus and KNX/IP drivers are present; a project naming another driver opens with that connection listed and refusing to start.



The I/O Configuration window on the reference works with the HillStation (IEC 60870-5-104), IntakeStation (DNP3) and TreatmentHall (BACnet/IP) connections open: HillDischargePressure chosen, address 12:4001:M_ME_NC, unit bar, and Read this tag answering 3.86 (Float) from the hill station's outstation

Driver settings

The **CONNECTION PARAMETERS** section changes with the driver. The settings you will meet most often:

Driver	Settings
Siemens S7	**IP address**, **CPU family**, **Rack**, **Slot**. An S7-300 is normally slot 2, an S7-1200/1500 slot 1
S7CommPlus	**IP address**, **Port**, **Username**, **PLC password**, **Timeout (ms)**, **Browse timeout (ms)**, **Allow writes**. No rack or slot
Modbus TCP	**IP address**, **Port** (502), **Unit / slave id**, **Framing** (**Modbus TCP** or **RTU over TCP (serial gateway)**)
Modbus RTU (serial)	**Serial port** (`COM3` , `/dev/ttyUSB0` , or better its `/dev/serial/by-id/...` name), **Baud rate** (9600), **Parity** (Even, the Modbus default), **Stop bits** (1), **Data bits** (always 8), **RS-485 direction** (Automatic for a USB adapter, or RTS for a converter that needs it), **Unit / slave id** (1–247), **Response timeout (ms)** (1000), **Retries** (1), **Inter-frame delay (ms)** (0: 3.5 character times, 1.75 ms above 19200 baud), **Inter-character timeout (ms)** (0: 3.5 character times plus 20 ms), **Adapter echoes what it sends** (off). Addresses are the Modbus TCP ones, optionally behind a unit: `40001` , `HR0` , `2:IR10` , `C5`
SunSpec	**IP address**, **Port** (502), **Unit ids** (`1` ; `1,2,3` for several devices behind one gateway), **Framing**, **Base addresses** (`40000,50000,0` , tried in order), **Request timeout (ms)** (3000), **Most models** (64) and **Most registers** (8192), past which a model chain is refused as malformed. A tag's address is `model/point` — `inverter/W` , `103/W` , `mppt/2/DCW` (MPPT module 2), `meter#2/W` (the second meter), `inverter/St.text` (the state's name) — with `unit:` in front when there are several units (`2:meter/W`). A point with a scale factor is a Double already scaled; **Browse Tags…** lists every point the device's models have, with units

Driver	Settings
M-Bus	**Connection** (**Serial level converter** or **TCP gateway**); for a converter **Serial port** (`COM3` , `/dev/ttyUSB0`), **Baud rate** (2400) and **Parity** (Even); for a gateway **IP address** and **Port** (10001, or whatever the gateway's raw TCP server uses); **Meters** (`1` ; `1,2,67290154` for several — primary addresses 0–250 and 8-digit secondary addresses, or 16 characters with the manufacturer, version and medium, `F` matching any digit), **Reply timeout (ms)** (1000; a gateway may need 2000), **Retries** (2), **Most telegrams per meter** (8) and **Initialise meters** (SND_NKE before the first read). A tag's address is `meter/record` — `5/energy` , `5/energy@storage1` (the reading at the last due date), `67290154/volume` , `5/date_time` , `5/meter.status` — or the record alone when the connection has one meter. Quantities are in Wh, J, m³, kg, W, °C, K or bar whatever the meter's display shows; dates are String tags in the meter's own local time. Each meter a poll names is read once per poll; a battery meter counts its readouts, so poll it every few minutes rather than every second. **Wireless M-Bus receiver**: **Serial port** is the IMST iU891A stick's (115200 baud is fixed), **Radio mode** (**C1 and T1**, or T1, C1 or S1 alone), **Meters** (identification numbers such as `12345678` , or blank for every meter heard) and **Expected interval (s)** (900); a tag is `12345678/volume` , plus `12345678/radio.rssi` and `radio.mode`
Allen-Bradley	**IP address**, **CPU slot**
Beckhoff	**IP address**, **AMS Net Id**
MELSEC	**IP address**, **Port**. An `X` address is hexadecimal, a `D` address decimal
Omron FINS	**IP address**, **Port** (9600), **Transport** (FINS/TCP or FINS/UDP), **Destination network** (0: local), **Destination node** (blank: over TCP the PLC says it, over UDP the last number of its IP address), **Destination unit** (0: the CPU), **Source node** (UDP only; blank: the last number of this computer's address), **Word order** (low word first, as CS/CJ/CP store a DINT or REAL, or high word first), **String byte order**, **Request timeout (ms)**, **Retries** (UDP). A Boolean tag on `T5` or `C12` reads the completion flag, a numeric one the present value. Over UDP the Ethernet unit must answer to the port the request came from (automatic, dynamic address conversion)
OPC UA	**Endpoint URL**, **Security mode**, **Security policy**, **Subscriptions**, **Publishing interval (ms)**, **Monitored-item queue**, **Trust server cert**, **Allow cleartext login**, **Username**, **Password**
MQTT PLC	**Broker host**, **Broker port**, **Client id**, **Base topic**, **QoS**, **TLS**, **Trust any server cert**, **Stale after (ms)**, **Username**, **Password**
MTConnect agent	**Agent URL** (the base address, without `/probe`), **Device** (name or UUID; blank for all), **Sample interval (ms)** (500), **Observations per request** (1000), **Request timeout (ms)** (5000), **Username**, **Password** (basic authentication). A tag's address is a data-item id from `/probe` ; **Browse Tags…** lists them
BACnet/IP	**Local interface** (0.0.0.0), **Local UDP port** (47808), **Broadcast address** (255.255.255.255), **BBMD address** (blank on the devices' own subnet), **Registration lifetime (s)** (300), **Device addresses** (blank; `instance=address` for a device that does not answer a broadcast, `/network/MAC` behind a router), **Request timeout (ms)** (3000), **Retries** (2), **Discovery wait (ms)** (3000), **Subscription lifetime (s)** (300), **Confirmed notifications**. A tag's address is `device:object-type:instance` , optionally with a property — `1201:analog-input:3` , `1201:analog-input:3:status-flags` ; **Browse Tags…** lists every point on every device that answers Who-Is. For BACnet/SC set **Data link** to **BACnet/SC (secure hub)**, then **Primary hub** (`wss://host:port`), **Failover hub**, **Operational certificate** (.pfx, or PEM with **Private key file**), **Certificate password**, **Issuer CA certificate**, **Check the hub's name** (on) and **Heartbeat (s)** (300); a device behind the hub may be named `instance=vmac:02AB34CD56EF` in **Device addresses**
IEC 60870-5-104	**IP address**, **Port** (2404), **Redundant address** (blank for none), **Common address** (1), **Common address size** (2 octets), **Object address size** (3), **Cause of transmission size** (2), **Originator address** (0), **k** (12), **w** (8), **t0**–**t3** (30, 15, 10, 20 s), **Interrogation interval (s)** (0: on connect only), **Counter interrogation interval (s)** (0: never), **Time tag zone** (UTC), **TLS**, **CA certificate file**, **Client certificate file**, **Client certificate password**. A tag's address is `commonAddress:ioa` , optionally with the expected type — `12:4001:M_ME_NC` ; **Browse Tags…** lists what a general interrogation reports

Driver	Settings
DNP3	**IP address**, **Port** (20000), **Master address** (1), **Outstation address** (10), **Event poll interval (ms)** (1000; 0 for unsolicited only), **Integrity poll interval (ms)** (0: at start-up, after a restart and after an overflow), **Unsolicited responses** (on), **Response timeout (ms)** (5000), **Keep-alive (ms)** (60000). A tag's address is `<group>:<index>` — `30:5` is analog input 5 — or `30v5:5` to read it as a float; **Browse Tags…** lists what the integrity poll reported
Tasmota	**Transport** (MQTT broker or HTTP), **Device topic** (the Topic in the device's MQTT settings), **Full topic** (`%prefix%/%topic%/`), **Broker host**, **Broker port**, **Client id** (blank: derived from this computer and the topic), **TLS**, **Trust any server cert**, **Username**, **Password** for the broker; **Device address** and **Web password** for HTTP; **Request timeout (ms)** (5000), **TelePeriod (s)** (300, to match the device), **Use device time** (off), **Device time zone**. A tag's address is a path in the device's JSON — `ENERGY/Power` , `AM2301/Temperature` , `Wifi/RSSI` — or `POWER1` for an output; **Browse Tags…** lists what the device reports to `Status 0` and, over MQTT, the outputs its discovery message names
Sparkplug B	**Broker host**, **Broker port**, **Client id** (blank: derived from this computer and the filters), **TLS**, **Trust any server cert**, **Username**, **Password**; **Group id** (`+` for every group), **Edge nodes** and **Devices** (an id, `+` , or a pattern with `*` and `?`), **Primary host id** (blank: listen only), **Request rebirth** (off), **Browse wait (s)** (5), **Stale after (s)** (0: off — Sparkplug reports by exception). A tag's address is `group/edge node/device/metric` — `PlantA/Line1/Press7/Count` — or `group/edge node//metric` for a metric of the edge node itself; a template member or an array element extends the metric, `…/Motor1/Speed` , `…/Levels[3]` . **Browse Tags…** lists every metric the births seen so far declare
Shelly	**Transport** (MQTT broker or HTTP), **Topic prefix** (the device's MQTT topic prefix — its device id unless changed; RPC over MQTT must be on), **Broker host**, **Broker port**, **Client id** (blank: derived from this computer and the prefix), **TLS**, **Trust any server cert**, **Username**, **Password** for the broker; **Device address** for HTTP; **Device password** (the device's own, user `admin` , answered by digest — blank when its authentication is off); **Request timeout (ms)** (5000), **Stale after (s)** (300; 0 for battery devices that sleep), **Use device time** (on). A tag's address is a component path in the device's status — `switch:0/apower` , `switch:0/aenergy/total` , `em:0/a_voltage` , `temperature:100/tC` , arrays numbered from 0 — or `switch:0/output` for an output; **Browse Tags…** lists everything `Shelly.GetStatus` reports, named from `Shelly.GetConfig`
KNX/IP	**Connection** (Tunnelling (UDP), Tunnelling (TCP) or Routing (multicast)), **Interface address** (tunnelling; blank and **Browse Tags…** searches the network), **Interface port** (3671), **Local interface** (0.0.0.0), **NAT mode** (off; UDP only), **Multicast group** (224.0.23.12) and **Individual address** (15.15.250) for routing, **Telegrams per second** (20), **Heartbeat (s)** (60), **Stale after (s)** (0: never), **Group address style** (3-level), **ETS group-address export** (an XML or CSV export, not the `.knxproj`), **Browse wait (s)** (10). A tag's address is `<group>@<datapoint type>` — `1/2/3@9.001` , `1/1/2@1.001` — with `;read` , `;read=<s>` , `;stale=<s>` and `;write=<group>` as options
LoRaWAN	**Network server** (ChirpStack v4 or The Things Stack v3), **Broker host**, **Broker port**, **Client id** (blank: derived from this computer and the application), **TLS**, **Trust any server cert**, **CA certificate file**, **Client certificate file**, **Client key file**, **Client certificate password**, **Username**, **Password** (on The Things Stack an API key; the username defaults to `<app>@<tenant>`); **Application id** (required: ChirpStack's application UUID, or The Things Stack's application id), **Tenant id** (The Things Stack only; `ttn` on The Things Network, blank on The Things Stack Open Source), **Devices** (blank: all; otherwise DevEUIs or device ids), **Expected uplink interval (s)** (3600; 0: never stale), **Per-device intervals** (`device=seconds` , comma separated), **Raw payload decoder** (None, or Cayenne LPP for uplinks the server left undecoded), **Browse wait (s)** (10). ChirpStack's MQTT integration must send JSON (`json=true`). A tag's address is `<device>/<path>` : the device by DevEUI (any case, with or without separators) or, on The Things Stack, by device id; the path into the codec's decoded object — `0004a30b001c0530/temperature` , `…/door/open` , `…/history/0` — or a link value: `_rssi` , `_snr` , `_fcnt` , `_fport` , `_gateways` , `_payload` (hex), `_battery` , `_margin` , `_external_power` . **Browse Tags…** lists the devices and paths heard since the connection subscribed

Driver	Settings
IO-Link	**Interface** (JSON Integration or ifm IoT Core), **Master address** (an IP address or host name, or `https://` for TLS), **Master number** (1; JSON Integration, for a modular gateway), **Username**, **Password** (HTTP basic for JSON Integration; IoT Core's own, user `administrator` when blank), **Certificate fingerprint** (the master's SHA-256, for a self-signed certificate), **Trust any server cert**, **Request timeout (ms)** (5000), **Process data layout** (`port1/Temperature=2:14:int*0.1; port1/OUT1=0:1:bool`), **IODD files** (`1=C:\IODD\sensor.zip`), **Expected devices** (`1=310:1234` ; blank: the IODD's, or the first seen), **Stale after (s)** (30), **Identity check (s)** (60), **Parameter refresh (s)** (60), **Read events** (on). A tag's address is `port<N>/pdin` , a field name such as `port1/Temperature` , `port1/pdin:2:14:int*0.1` , `port1/_status` or `port1/isdu/16:string`
Generic Cyclic UDP	**Local bind IP**, **Local UDP port**, **Expected payload size**, **Payload byte order**, **Validate CRC32**, **Receive timeout (ms)**
Generic Cyclic TCP	**Host**, **TCP port**, **Role** (connect out or accept), **Reconnect delay (ms)**, and the same payload settings

Hover over a field for its help text.

There is a filled-in example of every driver. The *reference works* demonstration carries one connection per shipped driver with its settings, groups and addresses done correctly — for example `RawWater` (Siemens S7, `192.168.10.11` , CPU family S7-1500, rack 0, slot 1), `Weigher` (Modbus TCP, `127.0.0.1` , port 502, unit 1), `IncomerMeters` (Modbus RTU, a `/dev/serial/by-id` USB adapter at 9600 8E1, units 1 and 2), `Scada` (OPC UA, `opc.tcp://127.0.0.1:4840` , subscriptions on) `PumpTestRig` (Generic Cyclic TCP, `127.0.0.1:34000`), `HillStation` (IEC 60870-5-104, `127.0.0.1:2404` , common address 12), `IntakeStation` (DNP3, `127.0.0.1:20000` , outstation 10) and `TreatmentHall` (BACnet/IP, device 2601 at `127.0.0.1:47809`). Only its Simulator connection is enabled, because the other controllers are not on your desk. Open it beside the form you are filling in. With **Also start simulated controllers for it** ticked in **Help ▸ Demonstration…**, six of them — the Modbus weigher, the cyclic test rig, the OPC UA server, the hill pumping station's IEC 104 outstation, the river intake's DNP3 outstation and the treatment hall's BACnet controller — are raised on `127.0.0.1` and switched on.

Several slaves on one RS-485 line. Give each slave its own Modbus RTU connection naming the same **Serial port**, or address the others from one connection with a unit prefix (`2:40001`). Either way there is one owner per port: requests from every connection on it are queued and sent one at a time with the specification's silence between them, because an RS-485 bus has one master and one conversation. Connections that share a port must agree on its speed, parity, stop bits and direction control; one that does not is refused with the settings already in use. A slave that stops answering is not asked again on every tag — its tags report the failure at once and it is retried after a backoff of 2 s doubling to 30 s — so one dead meter does not slow the others. Unplugging the USB adapter makes every connection on it Bad; it is reopened with a backoff when it comes back. A port carries one protocol at a time: while Modbus RTU connections hold a port — by any name for it, `/dev/ttyUSB0` or its `/dev/serial/by-id` link, `COM3` or `com3` — an M-Bus connection naming it does not open it and is Bad with the reason, *`/dev/ttyUSB0` is in use by the Modbus RTU connection 'Boiler room'*, and the same the other way round. The port passes to the other driver once the last connection holding it is stopped. An M-Bus connection has its port to itself — it has no queue to share it with another — so the meters on one M-Bus line are listed together in one connection's **Meters**; a second M-Bus connection on the same port is refused the same way. Modbus ASCII is not supported, and a device behind a serial-to-Ethernet gateway is read with the **Modbus TCP** driver's **RTU over TCP** framing instead.

OPC UA message security

The built-in OPC UA client signs, or signs and encrypts, every message itself — no add-on is needed, for security or for subscriptions.

Setting	What it does
Security policy	**Basic256Sha256** (what every server offers), **Aes128_Sha256_RsaOaep**, **Aes256_Sha256_RsaPss**, or **Most secure offered**. **None** sends everything in the clear and does not authenticate the server; the connection's log says so when it is made. The deprecated SHA-1 policies (Basic128Rsa15, Basic256) are never used.
Security mode	**Sign** (no message can be altered on the way) or **Sign and encrypt** (nor read). With the policy left at **None** and a mode chosen, the most secure policy the server offers for that mode is used.
Trust server cert	Lab use only. Accepts a server certificate nothing vouches for, and logs a warning each time. Off by default; leave it off.
Allow cleartext login	Lab use only, and only on policy **None**. A secured connection never needs it.

What happens when a secured connection is made:

1. The client asks the server for its endpoints over an unsecured discovery channel, the one every server answers, and picks the one the connection's policy and mode ask for. If there is none, the error names what the server does offer.
2. It checks the server's certificate before sending anything secret: in date, an RSA key of 2048 to 4096 bits, its application URI the server's own, and **trusted**. A certificate is trusted when the site PKI issued it (another Meelmo node's server), when it is in the folder `opcua/client-pki/trusted` under the state directory, or when a CA in that folder issued it.
3. A certificate that is not trusted is refused with its thumbprint, and written to `opcua/client-pki/rejected`. **To trust it, move that file into `trusted`** — the same move every OPC UA product asks for. The next connection attempt uses it; nothing restarts.
4. It opens the secured channel. The server has to answer with exactly the certificate its endpoint listed, and prove it holds the key; anything else is refused.
5. The session's password is encrypted with the server's certificate and the nonce the server just issued, so it is never on the wire in the clear and cannot be replayed.

The client's own certificate. On the machine that runs the site's certificate authority (a plant server, or a single-machine installation that created its site) the site PKI issues the client's application certificate, with its OPC UA application URI (`urn:<host>:Meelmo:OpcUaClient`), and renews it like every other site certificate; a server that trusts the site's root trusts it without a file being copied. Anywhere else the client makes itself a self-signed certificate on first use, kept in `opcua/client-pki/own` with its key sealed like every other secret on the node, and the log names the file. The server has to trust it: on most servers the first connection puts it in the server's rejected folder, from where an administrator moves it to trusted — on a Meelmo node's own server, in `opcua/pki/rejected` (see the *Industrial integration* chapter). Nothing is made for a node whose OPC UA connections all use policy **None**.

The secure channel is renewed at three quarters of the lifetime the server grants, with new keys each time. Tested in process against the node's own server under every policy and mode, and against node-opcua and the OPC Foundation's UA .NET Standard reference server; not yet against a PLC's built-in server.

OPC UA subscriptions

Polled, an OPC UA connection reads every tag every period, whether it changed or not. A large connection pays for that in time: eleven thousand tags in three groups at one second took the best part of half a minute a round, so every value was twenty or thirty seconds old when it was recorded, the historian had one sample a signal per round, and a group that overran its deadline was reported as *did not answer*

within … and reconnected. With **Subscriptions** on — the default — the server does the sampling instead: each tag becomes a monitored item sampled at its group's rate (or its own), and once a publishing interval the server sends only what changed, with its own source timestamp.

Setting	What it does
Subscriptions	**On** (the default) subscribes; **Off** polls, exactly as before. A connection saved without the setting subscribes too; one that chose **Off** keeps polling. A server that offers no subscriptions is polled instead, at each group's rate, and the log says *does not offer subscriptions … It is polled instead*.
Publishing interval (ms)	How often the server sends what its items collected. **0** follows the fastest group's rate.
Monitored-item queue	How many samples the server keeps for one tag between two publishes. A tag that changes faster than the publishing interval keeps every change up to this many.

What the connection does:

- **A tag holding still is recorded every period.** A server sends a value when it changes, and a tag that holds still sends nothing. So that trends, the historian and the tiles see what they saw when the connection was polled, the last value is recorded again, stamped with the time, whenever the tag's group period passes with nothing delivered. The server is asked nothing for it, and the value is the one the server last sent, so a timestamp tile's *last change* is still the last change.
- **Creating the items.** In batches of the server's own *MaxMonitoredItemsPerCall*, read when the session opens (1,000 a request when the server does not say); a server that refuses a batch as too large has it halved and the smaller size kept. Twelve thousand items are created in a few hundred milliseconds on a local network. Each item reports a change of status, value or source timestamp, so a value that holds steady is still recorded each time the server samples it fresh.
- **A tag the server will not monitor** — a node it does not know, an index range it will not take — is **Bad** with the server's reason as soon as the subscription is made, and costs the other tags nothing.
- **Values** are converted to the tag's data type exactly as a polled read would convert them, structure fields included, and carry the server's source timestamp.
- **A skipped message.** Every notification message is numbered. A number skipped is asked for again (*Republish*) before the message after it is delivered; one the server no longer has is counted as dropped on the next batch.
- **A lost link, session or server.** Every tag goes **Bad** with why, at once. The connection reconnects with a backoff of 1 s doubling to 30 s — each step of the connect bounded as always, so it never holds the node's start — and creates the subscription and its items again; the first values after it are marked as following a gap rather than carried on as if nothing had been missed. A server that answers nothing, not even a keep-alive, for two keep-alive periods is treated as a lost link.
- **Changing the tags** of a subscribed connection adds and deletes monitored items on the subscription already open; a tag in both keeps streaming without a gap.
- **Stopping** deletes the subscription on the server, waiting at most five seconds.

The connection's diagnostics show the subscription as the server revised it — publishing interval asked and granted, keep-alive and lifetime counts, the sampling intervals the items were given — with the number of items monitored and refused, publishes and keep-alives received, sequence gaps, messages republished or missing, Publish requests outstanding, and how many times the subscription was lost and made again. Tested in process against the node's own OPC UA server, under policy None and Sign and encrypt; not yet against a PLC's built-in server.

Acquisition modes

Each connection runs in one of three modes, and a driver offers only the modes it actually supports:

Mode	Who decides when a sample is taken	Use it for
Polling	Meelmo, on a timer (the group's poll rate)	Most fieldbus protocols
Subscription	The server (OPC UA, MQTT, an MTConnect agent, an IEC 104 or DNP3 outstation, a BACnet controller by change of value), when a value changes	Change-driven data, with the source timestamp preserved where the protocol carries one
Cyclic streaming	The controller, every cycle	Genuine sub-10 ms acquisition, down to about 1 ms

The mode follows from the driver and its settings: the MQTT PLC, MTConnect, IEC 60870-5-104 and DNP3 drivers subscribe, a Tasmota or Shelly connection subscribes over MQTT and polls over HTTP, a Sparkplug B or LoRaWAN connection subscribes, a KNX/IP connection subscribes to what the bus says (or, polled, answers from it), an IO-Link connection polls its master, the cyclic drivers stream, an OPC UA connection subscribes when its **Subscriptions** setting is on, and a BACnet/IP connection polls or subscribes as its mode is set. Projects saved before modes existed load as Polling.

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 guarantee 5 ms. Meelmo reports what it *achieved* in **Administration ▸ Diagnostics…** rather than repeating back what was asked for.

Writes

Writing to a controller is off everywhere by default and has to be allowed at three levels:

1. On the connection, **Allow writing to this connection**.
2. On the tag, **Allow writing to this tag**, optionally with an **Allowed range** (min and max, in engineering units).
3. For the person, a role that may command a tag (Operator, Engineer or Administrator).

A write is then made from the Signal Tree: right-click the tag and choose **Write value…**. Every write is recorded in the audit log. If a tag is writable but its connection is not, the tag's properties say so: *Writes are disabled on this connection, so every write to this tag is refused.*

A Tasmota device can be told to do far more than switch a relay — upgrade its firmware, reset itself, change its Wi-Fi — over the same topic. The driver sends only two commands: an output (`POWER1`–`POWER32`, as ON, OFF or TOGGLE) or `Dimmer` (0–100). A tag addressed to anything else is refused before anything is sent, and so is a device topic that is the default GroupTopic `tasmotas`, which every device in the group would obey. A write counts as done when the device confirms it; one it never confirms is reported as failed, with the output's state unknown.

A Shelly's RPC channel likewise reboots, updates, reconfigures and scripts the device. The driver makes only `Switch.Set` (true or false, 1 or 0, on or off) and `Switch.Toggle` (`toggle`) on a tag addressed `switch:<id>/output`; covers, lights and every other method are refused before anything is sent. A write counts as done when the device's RPC result confirms it; one it never answers is reported as failed.

A KNX bus obeys whoever sends a telegram: a device can be reprogrammed, its parameters changed and its individual address reassigned over the same interface. The KNX/IP driver sends only GroupValueWrite, and only to the tag's own group address or the one its `;write=` names — never 0/0/0, never to an individual address, never a property or memory write. A value the tag's datapoint type cannot carry (150 to a 5.001 percentage, 21.5 to a whole-number type) is refused before anything is sent. Over a tunnel a

write counts as done when the interface confirms it on the bus (L_Data.con); over routing there is no confirmation to wait for. KNXnet/IP itself authenticates nobody, so the write gate is the only gate: the interface belongs on a network only the building's systems reach.

A LoRaWAN connection never writes. A write would be a downlink — a command queued at the network server for a battery device in the field, delivered whenever it next opens a receive window, perhaps hours later — and the driver sends none, whatever **Allow writing to this connection** says: every write is refused before anything is published, and a connection that allows writes says in the log that it will refuse them. Send downlinks from the network server's own console or API.

An IO-Link connection never writes either. Its master would take process data out, ISDU parameters and port settings over the same interface — each a command to a sensor or actuator, or to the port it hangs on — and the driver sends none of them, whatever **Allow writing to this connection** says: every write is refused before anything is sent. Parameterise devices with the master's own tools, an IODD-based configuration tool or the fieldbus controller.

SunSpec marks two hundred points writable — trip curves, volt-var curves, reconnection and battery settings — and a wrong value in most of them changes how a grid-connected inverter protects the grid. The SunSpec driver writes only the active-power limit (`WMaxLimPct`, its enable and its revert and ramp timers, in model 123 or 704) and `Conn` in model 123, with the device's scale factor read just before so the number written means the percentage asked for. Scale factors are never written, and anything else is refused before anything is sent.

An M-Bus meter can be given a new primary address, a new baud rate, its clock set or its counters reset over the same bus. The M-Bus driver sends none of those. It sends SND_NKE, the selection of a secondary address and REQ_UD2 — link control and the request for data — and nothing else, so every write to an M-Bus tag is refused, whatever the connection allows, before anything is sent. On a wireless connection it sends nothing on the radio at all: it sets the receiver's link mode in RAM, listens, and keeps the last telegram of each meter. A meter not heard for twice its **Expected interval** turns Uncertain, saying when it was last heard; one never heard, or whose telegram is encrypted without a key in **Meter keys** (or with the wrong one), is Bad with the reason.

FINS can also switch an Omron PLC between RUN and PROGRAM, set its clock, write its files and force bits. The Omron FINS driver sends none of those: it sends Memory Area Read and Memory Area Write, and writes only the tag's own word or words, or for a bit address (`D100.05`) that one bit alone, without reading the word back. Timers and counters are never written, nor A0–A447, which the CPU alone writes; a tag addressed to either is refused before anything is sent. A write the PLC refuses is reported with its FINS end code in words — *the CPU is in PROGRAM mode*, *write-protected*.

A Sparkplug B connection is a host application, and three things it can do change somebody else's equipment, so all three are off until they are turned on. **Primary host id** makes it announce itself on `spBv1.0/STATE/<id>` — online when it connects, offline when it stops, with a retained Will for when it vanishes — and every edge node configured for that host rebirths, or holds its data, as it comes and goes. **Request rebirth**, together with **Allow writing to this connection**, lets it send `Node Control/Rebirth` to an edge node whose data it cannot place or whose messages it lost, at most once every thirty seconds; without it, that edge node's values stay Uncertain until it is born again on its own. And a write to a tag sends one NCMD or DCMD — only for a scalar metric the edge node's birth declared, never a `Node Control/` or `Device Control/` metric, and with no acknowledgement in Sparkplug: the value shows when the edge node publishes it.

Acquisition groups

Tags in a group are **read in one driver call and share one timestamp**. That is the unit of temporal coherence, and it is why grouping is a modelling decision rather than a convenience: **two values you intend to compare belong in the same group.** On the packaging line, `CaseWeight` and `WeightTarget`

are both in `Checkweigher`, so `WeightDeviation` subtracts two readings taken at the same instant.

Field	Meaning
Name	The group's name, and the middle part of every topic in it
Poll rate (ms)	How often the group is read. It applies to every tag in the group
Description	What the group is, in words, for the next engineer to read
Kind of asset	What kind of thing this group is — a filler, a checkweigher. Matched against the role catalogue so the commissioning check can say which of this kind's roles are not yet bound. Leave it blank when nobody has decided
Group enabled	Cleared, the group's tags are not acquired

Tags that are not in any group (*ungrouped*) are acquired in a bucket per poll rate, named for the rate: an ungrouped tag at 250 ms on `Line 1` has the topic `Line 1/r250ms/Length`.

Removing and renaming

Removing a group is not a local edit, and the editor asks. A group's tags are not deleted with it by default; they become ungrouped, so every signal id in the group changes — `Line 1/Fast/Length` becomes `Line 1/r250ms/Length` — and every screen binding, alarm rule, dataset column and report naming the old id stops resolving. Nothing reports that: asking the historian for a signal that does not exist returns an empty answer, which looks exactly like a machine that has not been running.

So × on a group shows the id on either side of the change and offers:

- **Delete the tags too** — removes them with the group;
- **Keep them, ungrouped** — keeps the tags under their new ids;
- **Cancel** — the default.

The same applies to renaming a connection, group or tag: the history recorded under the old topic stays under it. Rename before a line goes into production, not after.

Tags

Select a tag in the tree to edit it.

Field	Meaning
Name	The tag's name, the last part of its topic. In symbolic controllers this is often a path (`"MyDB".Axis.Speed`), which the Signal Tree opens into branches
Display name	What screens call the signal — a trend's legend, a value tile, a list's row, the web client's detail view. Optional: left empty, the signal is called by its name. It does not change the topic. See *Signal names* in *Signals*
Address	Where the value is in the controller, in the driver's own syntax: `DB1.DBD4` (S7), `40001` or `HR100` (Modbus), `N7:0` or a Logix tag name (Allen-Bradley), `ns=2;s=Line.Speed` (OPC UA), `D100` (MELSEC). Empty for Simulator tags
Data type	`Boolean`, `Int16`, `Int32`, `Int64`, `UInt16`, `UInt32`, `UInt64`, `Float`, `Double` or `String`, or **Auto** where the driver can decide the type from the address
Acquisition group	The group the tag is read in

Field	Meaning
Poll rate (ms)	Used only while the tag is ungrouped; *(the group's rate applies while this tag is in a group)*
Unit	The engineering unit, shown in the tree, on chart axes and in reports: `g`, `cases/min`, `%`, `bar`
Role	What this tag is to its asset, in the project's own words — for example `transformer.oilTemperature` — so a report written for a kind of machine finds it on any one of them. See *The semantic model*
Scaling	`raw ×` factor `+` offset: engineering value = raw × factor + offset. Leave factor `1` and offset `0` when the controller already delivers engineering units
Tag enabled	Cleared, the tag stays in the project but is not acquired
Read this tag	Connects and reads this one tag at its configured type
Allow writing to this tag / **Allowed range**	See *Writes* above

Auto data type

On most protocols the address already says what is at it: `DB1.DBX0.0` is a bit, `DB1.DBW4` a 16-bit word, a Modbus coil a bit, `N7:0` an Allen-Bradley integer file and `F8:10` a float file, and an S7CommPlus access path such as `lid:8A0E0001.A:Real` states its type. Choose **Auto** at the top of the **Data type** list and the tag takes its type from its address, and keeps taking it as the address is corrected.

Where the address fixes only the *width* — a 32-bit doubleword is a `DInt` or a `Real`, and S7 has no letter for the difference; a Modbus register is sixteen bits of whatever the device put there — the picker stays live and the choice stays yours, because a guess there returns a plausible wrong number rather than failing.

Where the address cannot say, the controller is asked: an EtherNet/IP read reply carries the tag's type, so an Auto Logix tag takes the controller's word when acquisition starts. A tag that did not ask for Auto keeps its configured type, and a disagreement is written to the log.

Whether or not Auto is used, an address and a type that contradict each other are named under the type picker and in validation. Such a tag would otherwise connect, acquire and publish a plausible number.

Read this tag

Read this tag connects and reads the tag being edited, and shows what came back and the type it was read at. Where the controller declares a type, it is compared with the configured one: a Logix `REAL` configured as `Int32` is named here, which matters because it does not fail — 1234.5 decoded as an integer arrives as 1150894080 and moves whenever the real value moves. **Test Connection** reads whichever tag is first; **Read this tag** reads the one you are commissioning. Disabled tags can be read too.

Simulator waveform

On a Simulator connection each tag has a **SIMULATOR WAVEFORM** section instead of an address:

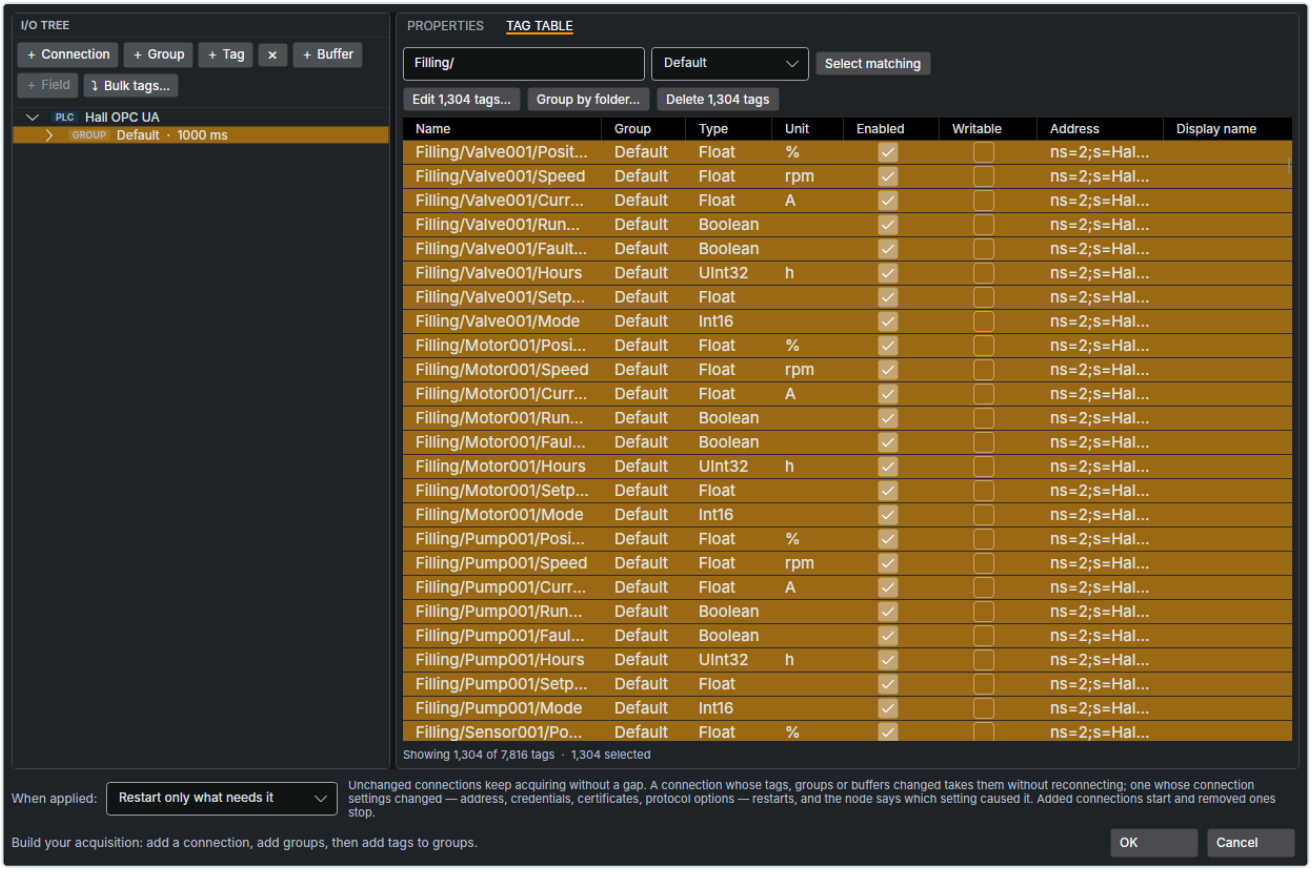
Field	Meaning
Wave	`Sine`, `Cosine`, `Ramp`, `Triangle`, `Step`, `Random`, `Constant` or `Boolean`
Base value	The centre (or start) value

Field	Meaning
Amplitude	How far it moves from the base value
Period (s)	The length of one cycle
Noise	Random noise added to every sample

The demonstration plants also use step sequences written into the project file (for example the `Running`, `Faulted`, `Blocked` and `Starved` states of each machine), which play a scripted timeline of values.

Working with many tags

A browse of an OPC UA server can bring thousands of tags into one connection — 7,816 on the site that asked for this — and editing them one at a time in the Properties pane is not organising them. The **Tag table** tab is where many tags are chosen and changed together.



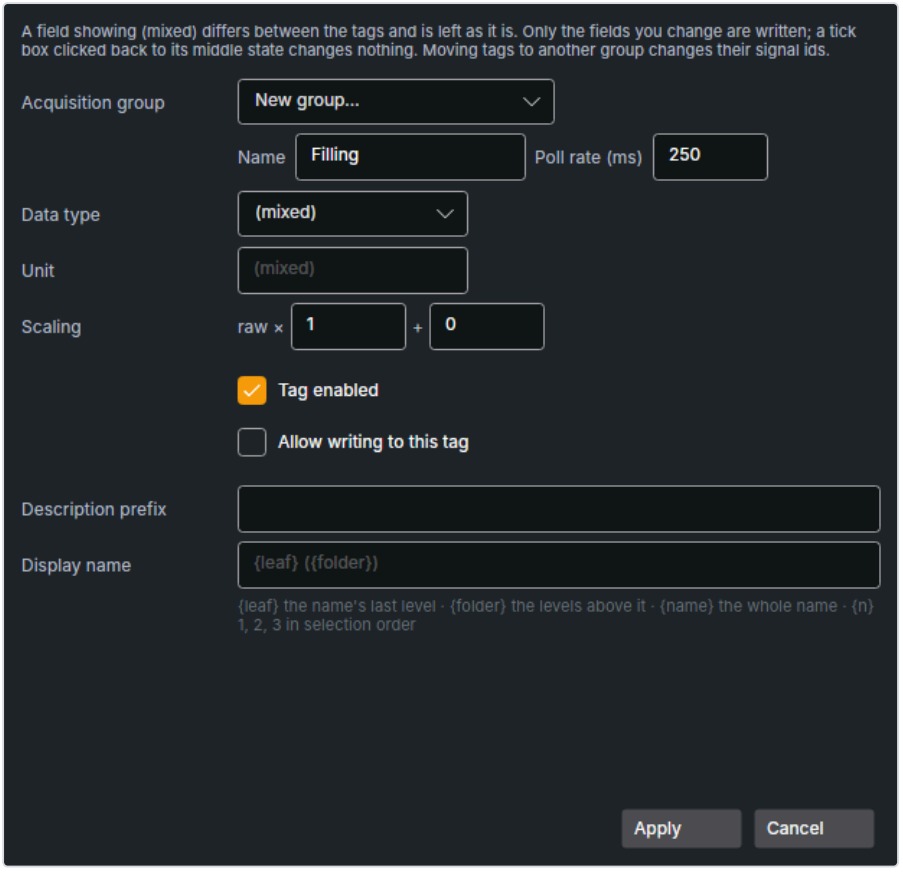
The Tag table on 7,816 tags browsed from an OPC UA server: the filter Filling/ and the group Default narrow it to 1,304 rows, all selected with Select matching; Edit 1,304 tags..., Group by folder... and Delete 1,304 tags above, and Showing 1,304 of 7,816 tags · 1,304 selected below

Choosing tags. The table lists the connection of whatever is selected in the tree, and choosing a group in the tree shows that group. Narrow it with the filter box — name, display name, address or description, in any case, or the unit exactly — and the group picker beside it. Select rows the way every list does: click, **Ctrl**-click, **Shift**-click, **Ctrl+A**, or **Select matching** (**Ctrl+Enter** in the filter) for every row the filter shows. A selection survives a change of filter wherever the new filter still shows it. Double-click a row, or press **Enter**, to open that tag in **Properties**.

Edit N tags... (**Ctrl+E**) opens one dialog for the whole selection:

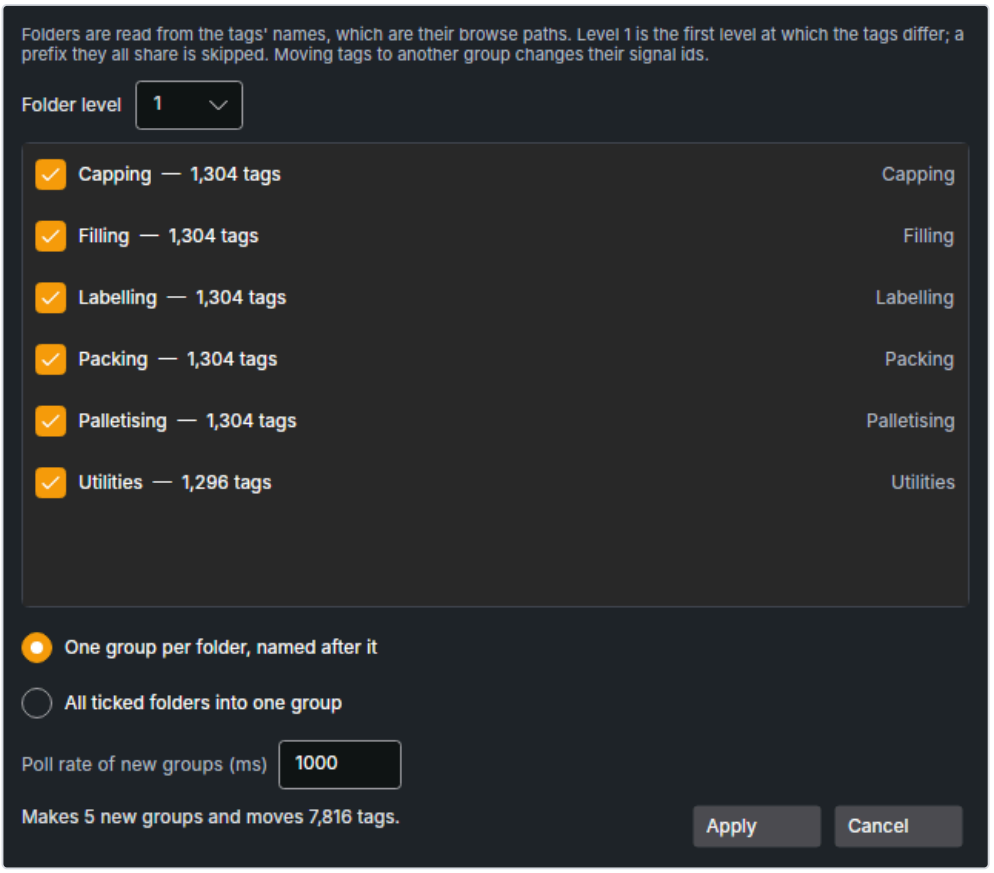
Field	What it does
Acquisition group	An existing group, *(ungrouped)*, or **New group…**, which asks for the new group's name and poll rate on the spot
Data type, **Unit**, **Scaling**	Written to every selected tag
Tag enabled, **Allow writing to this tag**	Three-state: ticked, cleared, or left in the middle to change nothing
Description prefix	Text put in front of every description
Display name	A pattern: `{leaf}` is the name's last level, `{folder}` the levels above it, `{name}` the whole name, `{n}` the tag's place in the selection. `{leaf} ({folder})` makes `Filling/Valve012/Position` *Position (Filling/Valve012)*

Only the fields you change are written. A field where the tags differ shows *(mixed)* and is left alone unless you change it, so moving 1,304 tags with six data types into a group does not give them all one type.



Edit 1,304 tags: a new group Filling at 250 ms; Data type and Unit showing (mixed); Scaling 1 and 0, shared by all of them; the display-name pattern's placeholder {leaf} ({folder})

Group by folder… (**Ctrl+G**) makes groups from the folders in the tags' names, which are their browse paths. It works on the selection, or on every row shown when nothing is selected. **Level 1** is the first level at which the tags differ — a prefix they all share, such as the server's root folder, is skipped. Choose **One group per folder, named after it**, which makes one group per folder at the poll rate you give and reuses a group that already has that name, or **All ticked folders into one group**, for one folder or several into one existing or new group. The dialog says what it will do before you press **Apply** — *Makes 6 new groups and moves 7,816 tags.* Two folders with the same name at that level, `Filling/Valve001` and `Capping/Valve001`, become `Filling.Valve001` and `Capping.Valve001`.



Group by folder on 7,816 tags: level 1, the six areas Filling, Capping, Labelling, Packing, Palletising and Utilities ticked with their tag counts, One group per folder chosen, and a poll rate of 1000 ms for the new groups

Delete N tags (**Delete** in the table) asks once, with the count.

None of these is applied to the node until **OK**. There is no undo inside the window; **Cancel** discards everything since it opened, and **OK** names whatever reads a tag that was moved or deleted before it applies — moving a tag to another group changes its signal id, exactly as editing its group in **Properties** does.

Removing or replacing a connection

When an edit removes a connection or a tag that something still reads, **OK** first asks *What reads the signals this change removes*: each computed signal, alarm rule, recording trigger, detection rule, screen, flow and report left naming one of them, and whether it reads only removed signals (*reads only …, so it will read nothing*) or still reads others. Replacing the demonstration's simulated **Packaging** connection with an OPC UA **Connection1**, for example, lists its computed signals and alarm rules. The choices:

- **Point them at the same-named tags** — offered when a removed tag has exactly one tag of the same name in the edited configuration (one under a connection you added, when there is a choice); the question lists each move, `Packaging/Line/Rate → Connection1/r1000/Rate`. Computed signals' expressions, scripts and model features, alarm conditions, the recording trigger and the detection rules are rewritten. Anything left over is asked about again.
- **Disable them** — switches the computed signals and alarm rules off and disarms the recording trigger.
- **Remove them** — deletes the computed signals and alarm rules (the recording trigger is disarmed).
- **Apply as it is** — applies the edit and leaves them; the node lists them as warnings in the activation's result and reports a computed signal that reads nothing as **Bad** with the reason.

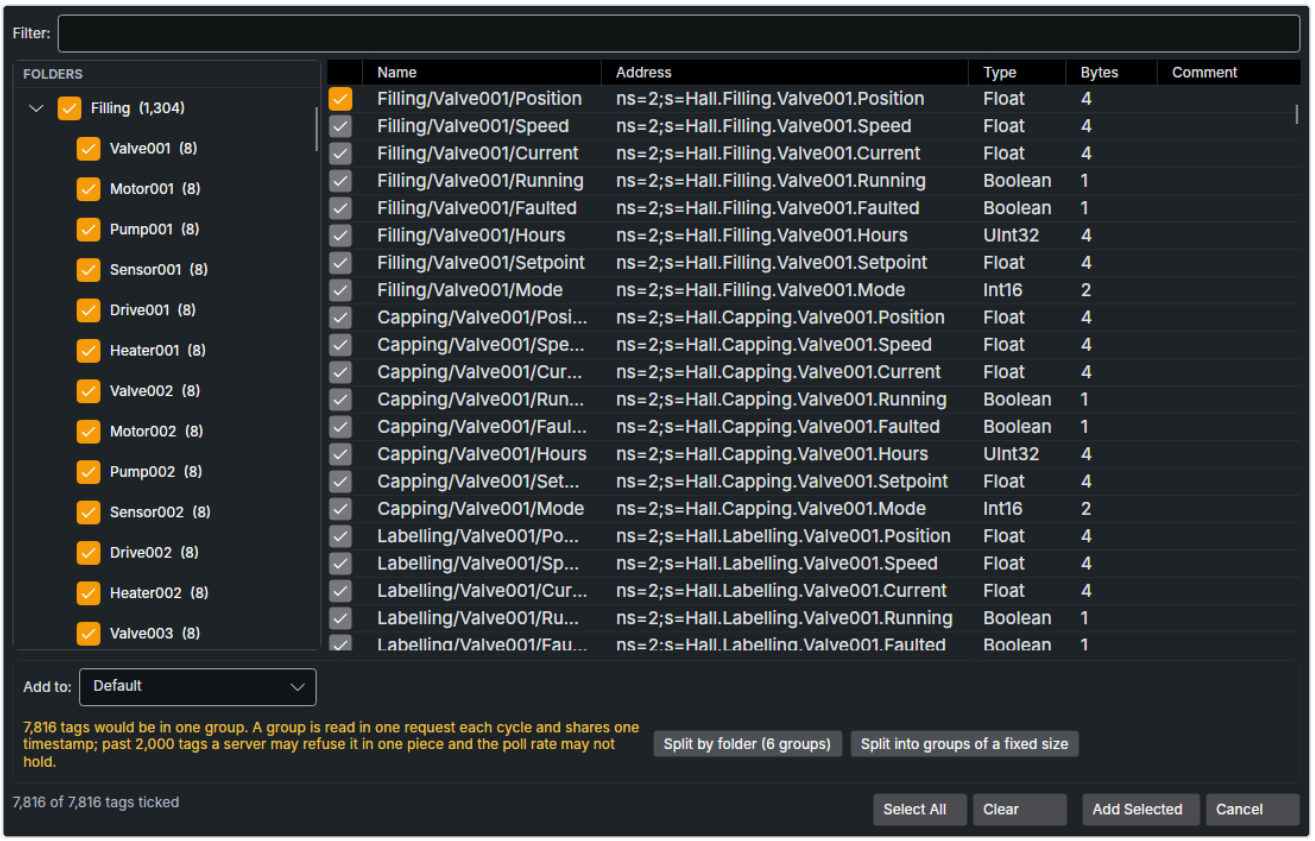
Screens, flows, datasets and reports are listed so they can be found; they are changed where they are edited. A reader named by its bare tag name (`[Rate]`) is not listed when a tag of that name remains, because it already reads it. **Cancel** returns to the window with nothing applied. After the edit, the removed connection's last values are shown as **Bad**, *Removed from the configuration in revision N*, wherever they were shown, and the connections you kept carry on with no gap.

Browsing a controller

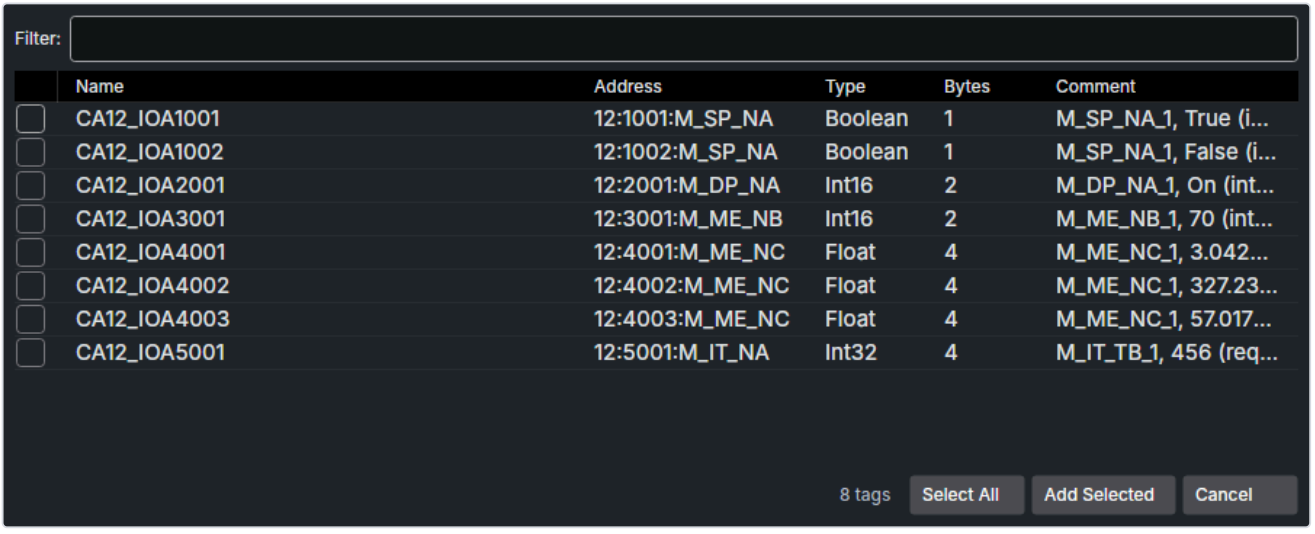
On Siemens S7CommPlus, Allen-Bradley, Beckhoff, OPC UA and MTConnect, addresses do not have to be typed — nor on an IEC 104 outstation, where **Browse Tags…** runs a general interrogation and lists every object reported, nor on a DNP3 outstation, where it lists every point the integrity poll reported, nor on a BACnet network, where it reads each device's object list and fills in each point's unit, nor on a Tasmota device, where it asks for `Status 0` and lists every reading and output, nor on a Sparkplug B namespace, where it lists every metric the birth certificates seen so far declare, waiting **Browse wait (s)** for more, nor on a Shelly, where it asks for `Shelly.GetStatus` and lists every value with the unit the API documents, nor on a KNX bus, where it lists the group addresses of the connection's ETS group-address export with their names and datapoint types, and every group address heard on the bus with a type guessed from its size, said to be a guess — with no interface set it searches the network and names the KNXnet/IP interfaces that answered — nor on a LoRaWAN application, where it lists every device and path heard since the connection subscribed, waiting **Browse wait (s)** for more — neither ChirpStack nor The Things Stack keeps an uplink on the broker, so a device appears only once it has sent, which for an hourly sensor can be an hour, nor on an IO-Link master, where it lists each port's status and mode, the vendor, device id, product and serial of the device on it, its process data, and every field of its layout or IODD with its unit and its value now:

1. Select the connection and press **Browse Tags…**.
2. The dialog reads the controller's symbol table (a large program reports progress and can be cancelled), maps the types, and walks into structures and function-block instances so individual members can be picked.
3. Type in **Filter by name or address** to narrow the list. The device is read once and cached; **Re-read** walks it again.
4. Tick the tags you want. Where the names are paths — an OPC UA server's folders, a program's blocks — the **Folders** tree on the left ticks a folder and everything under it, shows a folder that is partly ticked as neither, and narrows the list to the folder chosen. **Space** ticks the rows highlighted in the list; **Select All** and **Clear** tick and clear every row shown. The count of ticked tags is under the list.
5. Choose where they go in **Add to**: an existing group (the one selected in the tree is chosen to start with), *(ungrouped)*, **New group…** with its name and poll rate, **One new group per folder**, or **New groups of a fixed size**.
6. Press **Add Selected**.

An import that would put more than 2,000 tags into one group says so before anything is added: a group is read in one request each cycle and shares one timestamp, and past a couple of thousand tags a server may refuse the request in one piece and the poll rate may not hold. It offers **Split by folder** and **Split into groups of a fixed size**, and does neither on its own — a split decides every tag's signal id.



Browse Tags... on an OPC UA server's 7,816 tags, all ticked: the Folders tree with Filling (1,304) and its machines ticked, Add to Default, and the warning that 7,816 tags would be in one group with Split by folder (6 groups) and Split into groups of a fixed size beside it



Browse Tags... from an earlier build, before its folder tree and Add to row, on the reference works' HillStation connection, read from the simulated IEC 104 outstation by general interrogation: eight objects of common address 12 with their addresses, types, sizes and the values reported, a Filter box above, and Select All, Add Selected and Cancel below

Arrays expand into one entry per element, named by the index the controller's engineering tool uses — an `Array[1..10]` lists as `[1]` … `[10]` — because a tag holds one value.

On a SunSpec connection **Browse Tags…** lists every point of every model the device reported when it was discovered — model 160's trackers one by one, a second meter as `meter#2` — with the units and description from the SunSpec model, and an enum or bitfield twice: as its number, and as `.text` for its name.

On an M-Bus connection **Browse Tags…** reads each meter in **Meters** once — every telegram — and lists every record it sent, named as a tag addresses it, with the unit and a description (*Energy, storage 1*, *Flow temperature, maximum*), and the fixed header as `meter.id`, `meter.manufacturer`, `meter.medium`, `meter.status` and the rest. A meter that does not answer is left out and the browse says why. Manufacturer-specific data is listed as `manufacturer_data`, as hex. On a wireless connection nothing is asked: **Browse Tags…** lists the records of every meter heard since the receiver was opened, so wait for each meter's next telegram — a few minutes — before browsing.

An Omron CS, CJ or CP CPU keeps no symbol table that FINS can read — the names are in the CX-Programmer project — so on an Omron FINS connection **Browse Tags…** first asks which words to read: `D100-D199`, `CIO 0-9`, `E0_0-49`, `T0-31`, or a start alone for a hundred words. It lists each word with the value it held when read, as an unsigned 16-bit tag, to pick from and retype as needed. NJ/NX named variables are read over CIP, which this driver does not speak.

Adding many tags at once

⤵ **Bulk tags…** opens the **Add Tags** dialog. Every tab fills the same **PREVIEW** grid (**Name**, **Address**, **Type**, **Unit**), which you check before pressing **Add Tags**.

Tab	Use
Pattern	A run of tags from a template. `{i}` is the running address value and `{n}` a 1-based counter; formats like `{i:000}` are supported. Fields: **Name template**, **Address template**, **Start**, **Step**, **Count**, **Type**, **Unit**. Example: name `Motor_{n}_Speed`, address `DB1.DBD{i}`, start `0`, step `4`, count `40`
Paste list	Rows of `Name, Address, Type, Unit`, tab- or comma-separated, pasted from anywhere. A header row is detected. Press **Parse**
Excel	**Choose .xlsx…**: the first sheet is read, columns matched by header (Name, Address, Type, Unit) or by position
C# Script	C# that returns a sequence of tags built with `Tag(name, address, type, unit)`. Press **Run script**

Import from file… on the connection reads engineering exports directly (see *Importing a PLC tag list*).

Importing a PLC tag list

In the browser, **Administration ▸ I/O** has **Import tag list…** on every connection (see *Administration in the browser ▸ I/O*). It reads the tag list a control engineer exports from the PLC's engineering tool, shows every tag with the signal it would become and whether it can be imported, and adds the rows you tick to the connection as unsaved edits. **Save and activate** then applies them as one configuration revision. Signals already on the connection are matched by name; ticking one updates its address, type and description and keeps its identity.

In the Desktop, select the connection in **I/O Configuration** and press **Import from file…**. The same file is read the same way — the formats, addresses, names, types and reasons below apply to both — and the **Import a PLC tag list** window shows the preview before anything is added:

Part	What it shows
The line under the introduction	The file's name, the format it was read as, and how many rows are **New**, **Already here** and **Cannot import**
Namespace index	Only on an OPC UA connection. The namespace the PLC's tags are in (3 on a Siemens S7-1500's built-in server). Changing it reads the file again

Part	What it shows
Group for new signals	The acquisition group new tags go in; it starts at the group selected in the tree. A tag already here keeps its own group
Import	A tick per row. New rows start ticked; **Already here** starts unticked, because ticking it replaces that tag's address, type and description with the file's. A row that cannot be imported has no tick box, and stays in the list
Tag in the file	The tag's name, type and address as the export has them, to check the proposal against
Name, **Type**, **Address**, **Description**	The tag it would become, in the connection's driver's syntax
Status	**New** (green), **Already here** (amber; hover for what ticking it does), or **Cannot import** (grey) with the reason beside it — shortened to the column with an ellipsis when it is long; hover for the whole of it
Tick all new, **Untick all**, **Ticked: n of m**	Bulk ticks, and how many rows will be added of those that can be



The Import a PLC tag list window over the packaging line's Packaging connection, reading a TIA Portal PLC tag table: eight new filler, capper, checkweigher, labeller and air-pressure tags ticked with their S7 addresses, CaseWeight and WeightTarget already on the connection and left unticked, and a UDT_Recipe structure and a Byte status word listed as Cannot import with their reasons; Ticked: 8 of 10

Add to the connection adds the ticked rows to the project being edited; nothing reaches the node until **OK** and the project is applied, like any other edit in the window. A tag that is updated keeps its group, poll rate, unit and scaling.

The Desktop also reads the STEP 7 ASCII symbol variants (`.asc`, `.seq`) whose layout the shared reader does not recognise: when no header says what the columns are, they are inferred from their values, and the rows go into the same preview, addressed as written.

The files it reads. The format is decided from the content, not the extension:

Export	How to make it	What is read
TIA Portal PLC tag table	In the PLC tags editor, **Export** to `.xlsx` (the same columns saved again as `.csv` from Excel also read)	The **PLC Tags** sheet: Name, Data Type, Logical Address, Comment. Constants sheets are ignored
STEP 7 symbol file	Symbol Editor ▸ **Export**, `.sdf`	Symbol, address, type and comment
Studio 5000 / RSLogix 5000 tags export	Tools ▸ **Export Tags**, `.csv`	TAG and ALIAS rows (TYPE, SCOPE, NAME, DESCRIPTION, DATATYPE, SPECIFIER); COMMENT rows become the descriptions of array elements and timer members

Export	How to make it	What is read
Plain list	Any tool, or by hand	A header naming `name`, `address`, `type` and `description` (also `datatype`, `comment`, `nodeid`)
Studio 5000 project (`.L5X`)	File ▸ **Save As**, or **Export** a controller, as `.L5X`	Controller and program tags (`<Tag>` under `<Tags>`): name, data type with its dimensions, alias target and description. Read exactly as a tags export is: the same addresses, expansions and reasons
OPC UA NodeSet (`.xml`, often `*.NodeSet2.xml`)	The server's or the modelling tool's NodeSet export	Every variable (`UAVariable`): its node id as the address, its display name (or browse name) as the name, its data type — a built-in type, through the file's own aliases — and its description

A file larger than 8 MB is refused. The two XML exports are read as the uploads they are in the browser: a file that declares a document type (`<!DOCTYPE …>`) is refused without being read, since no export writes one and entity declarations are how an XML file makes its reader expand or fetch things; nothing named in a file is ever fetched; and XML nested more than 64 elements deep is refused as unreadable — a real project is about a dozen deep.

The address, per driver. A tag list is translated into the address syntax of the connection's driver:

Driver	Siemens (TIA Portal, STEP 7)	Rockwell (Studio 5000)	Plain list
Siemens S7	The absolute address, `%MW10` → `MW10`, `%DB10.DBD4` → `DB10.DBD4`; German mnemonics `%E1.3` → `I1.3`, `%AW12` → `QW12`	Cannot be imported	As written, with `%` and padding removed
S7CommPlus	The quoted symbol, `"Motor_Run"`	Cannot be imported	As written; the name when blank
Allen-Bradley (EtherNet/IP)	Cannot be imported	The tag name; `Program:MainProgram.Step` for a program tag	As written; the name when blank
OPC UA	`ns=3;s="Motor_Run"`, in the **Namespace index** you set	`ns=3;s=Line_Speed`, `ns=3;s=Program:MainProgram.Step`	As written
Simulator	No address	No address	No address

A NodeSet's node ids are the address on an **OPC UA** connection, exactly as the file writes them (the **Namespace index** does not change them), and the simulator needs none; on any other connection each row is refused as *The driver cannot read this controller's tags*. Other drivers take only a plain list. The OPC UA node ids follow a Siemens S7-1500's built-in server; a gateway in front of a Logix controller may prefix its own path, so read one tag with **Read this tag** before importing many.

Names and types. The signal's name is the tag's name with TIA Portal's quotes removed and `/` or a control character replaced by `_` (the topic separator), cut to 128 characters. A program tag whose name another tag in the file also has is named for its program, `MainProgram.Counter`. The comment becomes the signal's **Description**. Types map by width: `Bool` → `Boolean`; `Int`, `UInt`, `Word` → `Int16`; `DInt`, `UDInt`, `DWord`, `Time`, `Date`, `TOD` → `Int32`; `LInt`, `LWord`, `LTime` → `Int64`; `Real` → `Float`; `LReal` → `Double`; `String` → `String`. Unsigned types take the signed type of their width, as a browse does, so a `UDInt` above 2 147 483 647 reads negative.

Expanded. A Studio 5000 array of a scalar up to 256 elements becomes one signal per element (`Zone_Temps[0]` … `Zone_Temps[3]`). A `TIMER` becomes `.PRE`, `.ACC`, `.EN`, `.TT`, `.DN`; a `COUNTER` `.PRE`, `.ACC`, `.CU`, `.CD`, `.DN`, `.OV`, `.UN`. An alias takes the type of the tag it points at, or `Boolean` when it points

at a bit (`Local:1:I.Data.0`).

Cannot be imported, with the reason shown beside the row:

Reason	Example
A structure, with no single value to read	a UDT such as `Recipe_UDT` ; its members are in the project, not in a tags export — use **Browse Tags…**; a NodeSet variable whose data type the file defines (`ns=1;i=3001`)
No signal type for the type	`S5Time` , `DTL` , a STEP 7 `TIMER` ; a NodeSet `DateTime` , `Guid` or `LocalizedText`
8 bits, which the driver would read as 16	`Byte` , `SInt` , `USInt` , `Char` (accepted on OPC UA and the simulator)
The array is not expanded	a Siemens `Array[..] of` , a Logix array of more than one dimension or more than 256 elements, a NodeSet variable with a value rank of 0 or more
No address for the driver	a tag without an absolute address, on the S7 driver or in a plain list without one
Not an address the driver reads	a peripheral access `%IW256:P` , `PEW 256`
Not as wide as the tag's type	`LReal` at `%MD120` , `Int` at `%MD30`
A block, not a value	`DB 10` , `OB 1` in a symbol file
An alias whose type the export does not say	an alias of an I/O module's structure
The driver cannot read this controller's tags	a Studio 5000 export on an S7 connection
An earlier row already proposes the name	two tags that become the same signal name

Circular buffers

Some controllers produce data faster than anything can poll them and publish it as a **ring**: fifty pressure samples taken every 20 ms and overwritten in turn, or a queue of the last ten pieces that passed a weigher. Polled as an ordinary tag, such a buffer yields one record in fifty, chosen by the network.

+ Buffer on a connection reads the whole ring instead. The drain reads the pointer, works out which records are new, reads them, and publishes each record as its own sample on an ordinary tag, with its own timestamp. A buffer of fifty samples is still **one** tag to trend, alarm on, record and historize, and the trend has the cadence the controller saw rather than the one the poll rate left.

A buffer contains no protocol: every address comes from an **Address pattern** with `{index}` where the number goes (`DB100.DBW{index}` , `N7:{index}` , `HR{index}` , `ns=2;s=Samples[{index}]`), and every read goes through the connection's own driver. **Any driver that can read a tag can feed a buffer.**

Buffer fields

Select a buffer to edit it. The section **CIRCULAR BUFFER** explains: *read as a buffer it yields all of them, each with its own timestamp, on one tag per field.*

Section	Field	Meaning
	Name, **Kind**, **Poll rate (ms)**, **Buffer enabled**	**Kind** is `Stream` (samples on a clock) or `Events` (occurrences with no cadence). The poll rate is how often the ring is drained
The ring	**Records**	How many records the ring holds
	Stride	The distance from one record to the next, counted in whatever the address pattern counts (`DB100.DBW{index}` counts bytes, `N7:{index}` counts words)
	Record 0 at	The number that addresses the first record
	Address pattern	The pattern with `{index}`
The newest record	**Found by**	`Pointer` (a word naming a record), `WriteCount` (a counter of every record ever written) or `RecordSequence` (a number carried in each record)
	Pointer, **Pointer names**, **Value for record 0**	The pointer's address; whether it names the last written or the next record; the pointer value that means record 0 (for example `1` where positions are numbered 1…10)
	Sequence field	For `RecordSequence`, the field holding the number
	Write counter	Optional beside a pointer, and the only thing that makes loss detectable
A read the controller did not write through	**Counter above**, **Counter below**, **Read attempts**	Integrity counters read before and after the records; a mismatch means a torn read, which is discarded and taken again (default 3 attempts). Configure both or neither; pointing both at the pointer guards a buffer with no counters of its own
When a record happened	**Time field**, **One count is**, **Counted from**	Which field holds a time, its unit (milliseconds, seconds, microseconds, ticks, minutes, days or custom) and what it counts from: nothing (use the cadence), the Unix epoch, midnight, the .NET epoch, a fixed instant, an origin address in the controller, or the newest record (relative only)
	Origin instant (UTC), **Origin address**, **Read that as**	The origin for elapsed counts
	Counter wraps at	For a counter that rolls over, such as 65536
	Controller clock (min ahead of UTC)	For a controller whose clock runs in local time (120 for Central European summer time)
When the records carry no time of their own	**Sampling interval (ms)** (Stream), **Record spacing (ms)** (Events)	The cadence used to place records in time
Draining	**Most per drain**	The most records read in one drain
	Take what is already in the ring on start	Off is right for a stream with no time field, because old samples stamped as if just arrived are fabricated history. On is right for a queue of pieces not yet read
	Drain once	Reads the buffer now through the real driver and shows the records that came back — the check **Test Connection** cannot make, because a tag a buffer feeds has no address of its own

Select a field (under the buffer, added with **+ Field**) to edit it: **Name**, **Feeds tag** (the tag that receives the value), **Offset in record**, **Address pattern** (blank inherits the buffer's) and, for a field that feeds no tag, **Data type**. A field takes its data type, unit and scaling from the tag it feeds. A field feeding no tag is read and used but never published, which is how a record's own sequence or timestamp reaches the drain.

Records overwritten before a drain reached them are counted as dropped, not quietly absent — except under a bare pointer, which cannot tell a standstill from a full lap and so reports no losses.

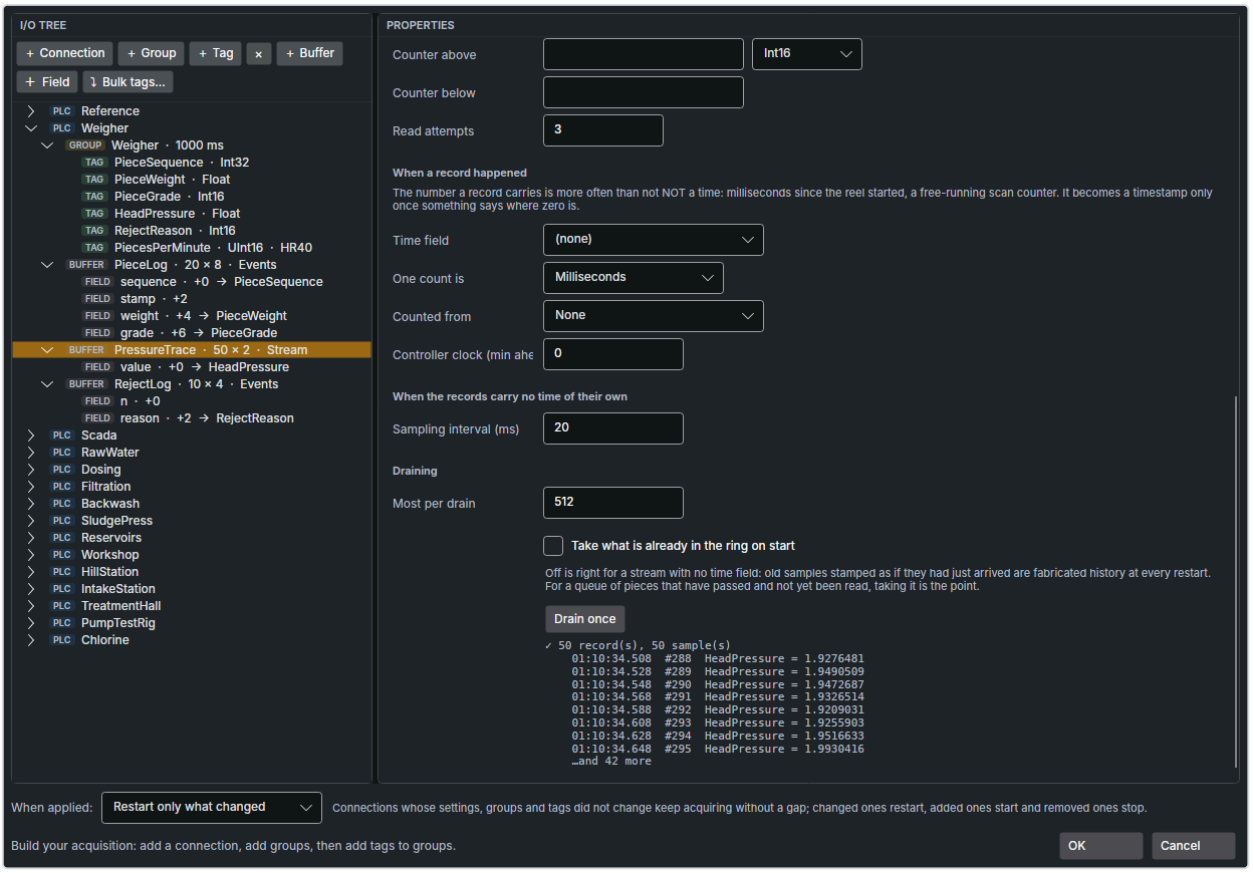
Worked examples from the reference works

The reference works' `Weigher` connection (Modbus TCP) carries one buffer of each anchor:

Buffer	Kind	Found by	Ring	Fields → tags
`PieceLog`	Events	Pointer at `HR900`, write counter at `HR902`, integrity counters at `HR904` / `HR905`	20 records, stride 8, from `HR0`	`sequence` → `PieceSequence`, `stamp` (Int32, ms since midnight, not published), `weight` → `PieceWeight`, `grade` → `PieceGrade`
`PressureTrace`	Stream	Write counter at `HR910`	50 records, stride 2, from `HR200`, sampled every 20 ms	`value` → `HeadPressure`
`RejectLog`	Events	Record sequence field `n`	10 records, stride 4, from `HR400`	`n` (not published), `reason` → `RejectReason`

To watch them harvested:

1. **Help ▸ Demonstration…**, select **Reference works (simulated)**, tick **Also start simulated controllers for it**, press **Open**.
2. **Acquisition ▸ I/O Configuration…**, select the `PressureTrace` buffer under `Weigher`, press **Drain once**. The records come back with their timestamps 20 ms apart.
3. Close with **OK**, press `F5`, and plot `HeadPressure`: fifty samples arrive per second although the ring is drained only twice a second.



The PressureTrace buffer under the Weigher connection selected in I/O Configuration, with the simulated controllers running: Stream, WriteCount, a sampling interval of 20 ms and Take what is already in the ring on start cleared; Drain once has read 50 records, each HeadPressure sample 20 ms after the one before

Lineage: what a tag feeds

Acquisition ▸ Lineage… shows where any signal comes from and everything that depends on it, computed from the configuration the node is running at the moment you ask — never from a list somebody keeps beside it.

The chain it draws, left to right, is the one data flows along:

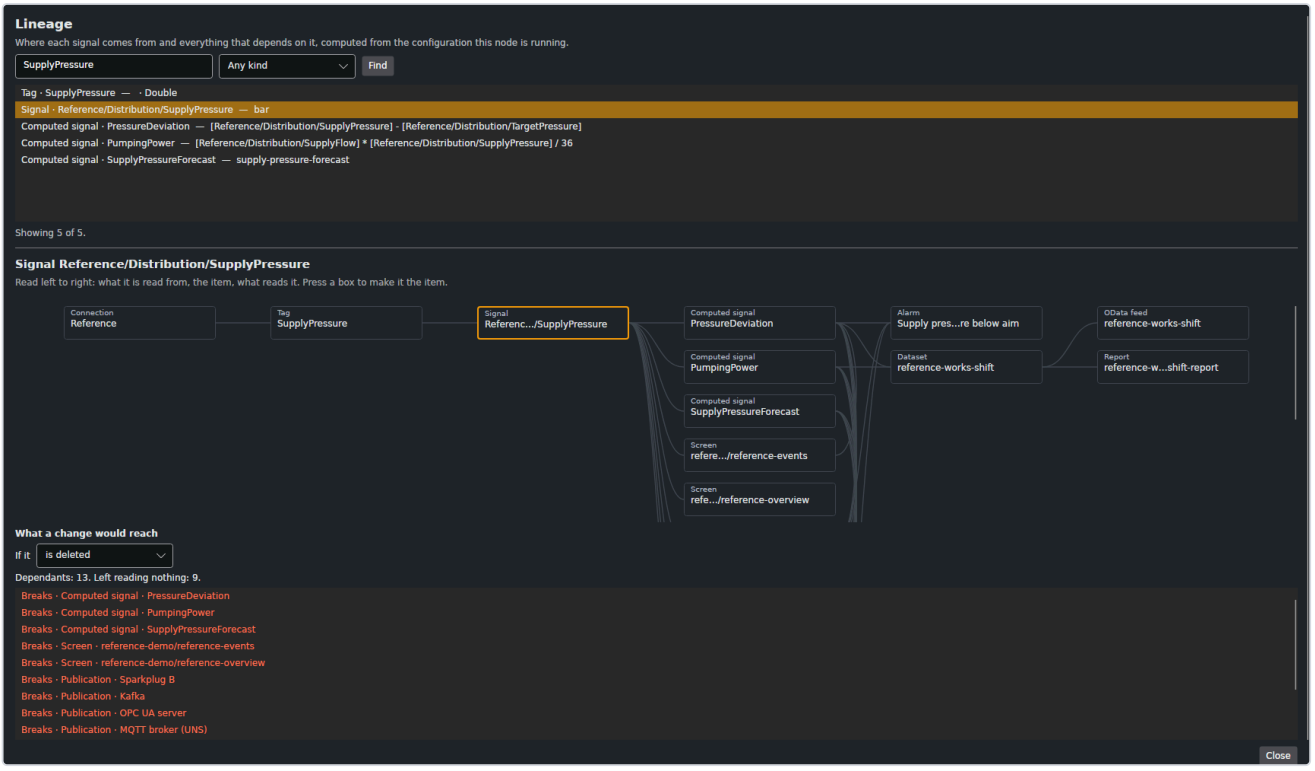
From	To	Found by
Connection	its tags	the connection holds them
Tag	its signal	the topic acquisition publishes, `Connection/Group/Tag` (or `Connection/r250ms/Tag` when ungrouped)
Signal	computed signal, alarm, recording trigger	the expression names it — as `[Line 1/Fast/Speed]` or as the bare tag name `Speed`, resolved exactly as the engines resolve it
Signal	model signal	one of its features names it, by topic or by tag name; the edge shows the feature, such as `mean(Reference/Distribution/SupplyPressure,60)`
Signal	C# script signal	the script calls `Tag("…")` with it as a string literal; the edge shows the call
Signal	detection rule, model, flow, screen, dataset	the document names its signal id, a `[topic]` in its text, or an expression or signal field that resolves to it
Alarm	flow	a flow's *rule* setting names it

From	To	Found by
Dataset	report, dataset drop, OData feed	a report's data source names it; every saved dataset is an OData entity set
Signal, alarm	MQTT, Sparkplug B, Kafka, OPC UA server	the publication settings: a sink publishes every signal, under the topic shown on the edge
Signal	MQTT broker (UNS)	the node's own broker is on with **Publish this node's signals**: it publishes every signal into the namespace, and alarms not at all

A C# script is read for the names it states, not run: `Tag("Reference/Intake/PumpRunning")` is drawn, but a name the script puts together while it runs — `Tag(line + "/Speed")`, an interpolated string, a variable — cannot be known from the configuration and is not drawn. A script that should show its inputs here names them as literals.

The topic on an edge into the MQTT broker is the one a signal is published on when the asset model places it nowhere else: the namespace's enterprise level, then the signal's own path with the root path taken off. A signal the asset model places is published under its asset's path instead, which the broker reads from the asset model as it runs.

Search by name, address or kind, then double-click an item. Each box in the drawing is a button: press one to make it the item. A column with more than 24 boxes says how many more it holds.



Acquisition ▸ Lineage on the reference works: SupplyPressure found by name; its drawing from the Reference connection and its tag to the two computed signals, the screen, the alarm, the dataset, its OData feed and the shift report that read it; and What a change would reach for a deletion, three dependants left reading nothing

For a connection, a tag or a signal, **What a change would reach** lists every dependant for the change you pick — deleted, renamed, retyped, moved or switched off — nearest first:

- **Breaks** — it reads the item directly and would read nothing afterwards. A rename breaks what names the old topic, and leaves working what names the tag by its bare name.

- **Changes** — it goes on working on something that changed under it: a new type or address, or a value that now arrives bad from a computed signal upstream.

A lineage is bounded so that a large plant costs a bounded amount to describe: the graph holds at most 200,000 items and a walk at most 32 steps and 5,000 items, and either says when it stopped short.

A tag that exists in the controller but is not configured has no place in a graph computed from the configuration. The browser's **Lineage** page can add them for one connection on request, from a browse of the device — see *Reports and analysis in the browser ▸ Lineage*.

Before a destructive change is applied

The I/O Configuration window asks before anything downstream is broken:

- × on a connection names, under the question, what would read nothing once it is gone. × on a tag that something reads asks too, with the same list; a tag nothing reads is removed without a question, as before.
- **OK** compares the edited project with the one the node is running. When any connection or tag that something depends on is deleted, switched off, renamed, retyped or moved, **What this change reaches** lists each change and its dependants and asks *Apply anyway?*

The Signal Tree's **Delete connection…** and **Delete tag…** ask the same question with the same list, and **Disable connection** asks **What this change reaches** when something reads the connection. A tag deleted from the tree is always asked about, even when nothing reads it.

The browser's **Administration ▸ I/O** asks the same question before **Save and activate** — see *Administration in the browser ▸ I/O*.

Tag drift

A controller is reprogrammed while the node's configuration still describes the old program: a tag is renamed in the PLC, a data block is reorganised and every offset moves, a `REAL` becomes an `LREAL`. Acquisition goes on reading — sometimes the wrong word, sometimes nothing — and nothing else says so. The drift check does.

The node browses the device again, compares the tag table with the **baseline** — the first table it read, or the one somebody last accepted — and reports:

Change	Means
Added	in the device now, not in the baseline
Removed	in the baseline, gone from the device
Retyped	the same address with a different data type
Moved	the same name at a different address — the change most likely to leave a configured tag reading the wrong word without an error
Renamed	the same address under a different name

Differences that touch a tag this connection is configured to read are listed first, with the tag named.

Which devices can be checked: the drivers that can list a device's tags — **OPC UA**, **Siemens S7CommPlus** (S7-1200/1500), **Allen-Bradley** Logix (not PLC-5, SLC 500 or MicroLogix, which have no symbol table), **Beckhoff ADS**, **BACnet/IP**, **MTConnect**, **DNP3** and **IEC 60870-5-104**. Modbus, classic S7, MELSEC, the MQTT PLC bridge, cyclic UDP and the simulator have nothing to browse, and the review says *Cannot be browsed*.

Setting it up

In **Acquisition ▸ I/O Configuration…**, on the connection, **TAG DRIFT**:

Field	Meaning
Check every (minutes, 0 = on demand)	How often the node browses the device again. `0`, the default, checks only when somebody asks. Anything shorter than 15 minutes is read as 15; the longest is a week
Send a drift through the alarm notifications	When a check finds a drift it has not reported before, it is sent once through the node's alarm channels — mail, webhook, Slack, Teams, PagerDuty, Opsgenie — as *Tag drift on {connection}*, at **Alarm** severity when it touches a configured tag and **Warning** otherwise, subject to the minimum severity those channels are set to

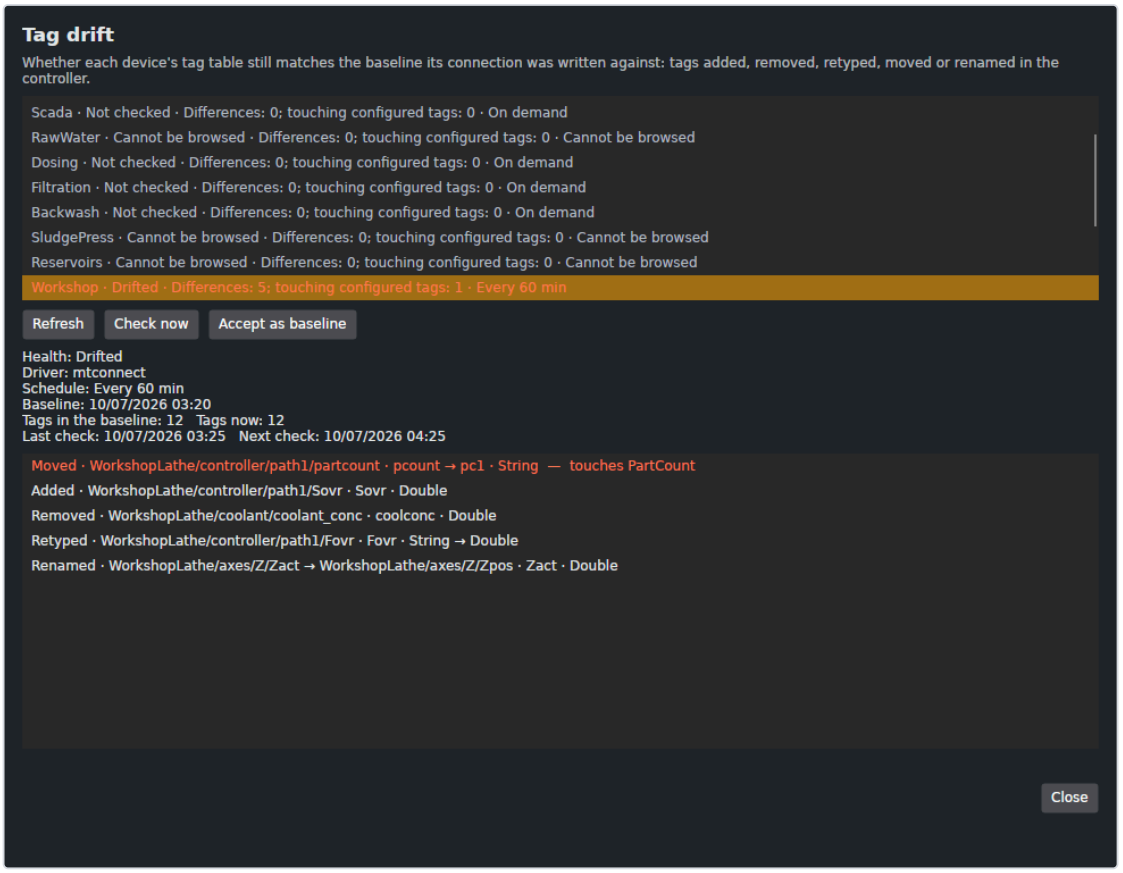
Changing either restarts no connection: the check is not acquisition's.

How it treats the device

- **On a session of its own.** Every browse opens its own driver session, lists the tags and closes it. The session acquisition reads through is never used, so a slow or refused browse is a failed check and never a gap in the line. On a controller with few connection slots it does take one, briefly.
- **One at a time.** One browse runs at a time on the whole node, each abandoned after two minutes, and a connection already being browsed is not browsed again until it finishes.
- **Backing off.** A device that refuses is asked half as often after each refusal — 60, 120, 240 minutes — up to once a day, until it answers. A driver that cannot browse is not asked again until the connection's driver or settings change.

The review

Acquisition ▸ Tag drift… lists every connection with its state — **Matches**, **Drifted**, **Browse failing**, **Not checked** or **Cannot be browsed** — the number of differences and the schedule. Select one to see the baseline, when it was taken and by whom, the latest browse and every difference.



Acquisition ▸ Tag drift with the workshop lathe chosen after a control update: Drifted, five differences, one touching a configured tag — the part counter moved from pcount to pc1 under the PartCount signal, the spindle override added, the coolant concentration removed, the feed override retyped from String to Double and the Z axis renamed Zact to Zpos

- **Check now** browses the device at once, whatever its schedule.
- **Accept as baseline** makes the latest browse the baseline, after asking. The node keeps who accepted it beside the baseline (the last 20 acceptances are listed), and this installation's audit trail records `AcceptDrift`.

On a workstation joined to a node, both are done at the node as the person signed in at the workstation, not with the workstation's own data account: the node checks their permission to change I/O, keeps their name beside the baseline and records `drift.check` and `drift.accept` in its trail. Someone without that permission is told so.

The baselines are kept beside the node in `drift.json`, so they survive a restart and a change of the process holding the node. A device listing more than 50,000 tags is compared on its first 50,000, and the review says so.

Each connection's count of unaccepted differences is also a signal, `Health/Drift/{connection}`, published down the same path as the acquisition's own timing: it trends, records and can carry an ordinary alarm rule, for example `[Health/Drift/Packaging] > 0`. It is bad quality while the device cannot be browsed.

Editions. Checking on demand and accepting are in every edition. Scheduled checks are part of the Standard and Professional editions; the Community Edition checks on demand only, so its devices are never browsed by a clock nobody chose.

Starting and stopping acquisition

Action	How
Start	**Acquisition ▸ Start**, F5, or the triangle on the toolbar
Stop	**Acquisition ▸ Stop**, F6, or the square on the toolbar

While running, the toolbar pill reads **ACQUIRING** with the samples per second, the status bar shows the driver load, and the Event Log records each connection coming up or going down.

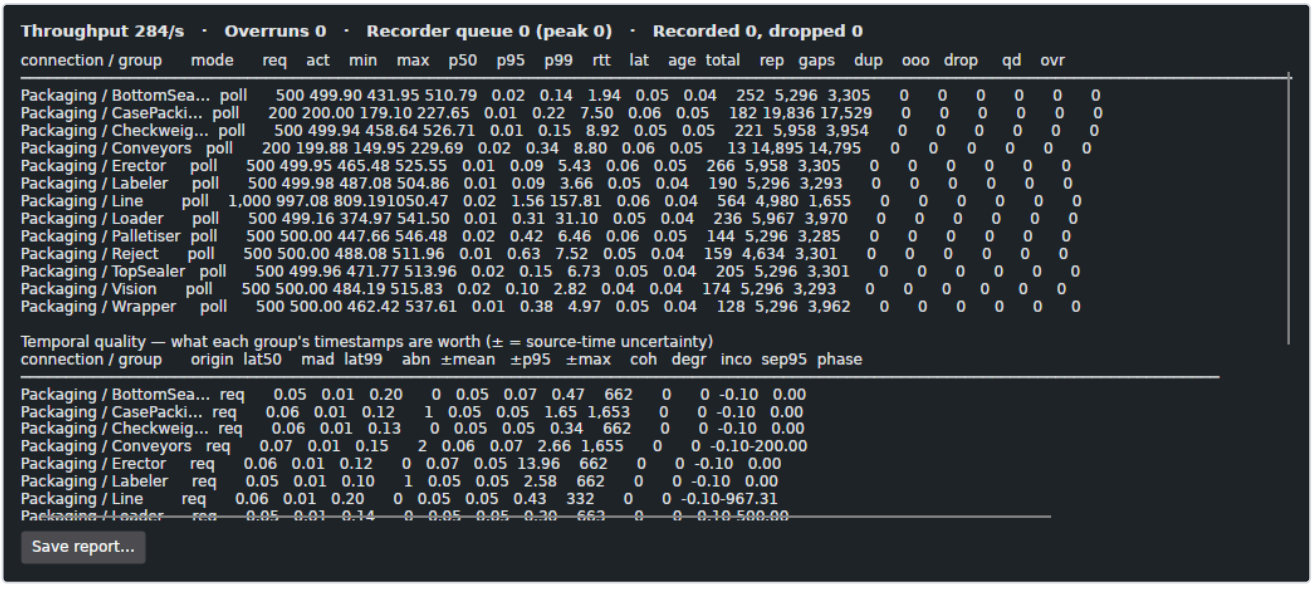
On a client attached to a node, Start and Stop act on the node's acquisition, which keeps running when the Desktop is closed. On a standalone workstation, acquisition stops when the application closes.

When a connection fails, the others keep running. Its tags go to bad quality (a red dot in the Signal Tree), the driver keeps retrying with a backoff, and the Event Log records the loss and the recovery. Quality is carried end to end: a bad reading is never silently plotted or recorded as a good number.

When Start refuses. Start is refused when the project enables more tags than the licence allows (the demonstration window warns about this beforehand) or when the account may not control acquisition. The reason is shown.

Acquisition diagnostics and timing

Administration ▸ Diagnostics… is the screen that proves the commissioning. Open it before you believe a trend: a line that is not sampling the way it was configured is either found here, at commissioning, or found months later during an investigation, when the recording is already wrong.



The Acquisition Diagnostics window while the packaging line acquires: about 284 samples a second, one row per acquisition group with its mode, requested and actual period, latency percentiles and sample counts, then the temporal-quality table and the coherence frames

The header line gives the total throughput, poll overruns, the recorder's queue (and its peak) and the recorded and dropped sample counts. Below it is one row per connection and group, refreshed live:

Column	Meaning
connection / group	Which group the row describes

Column	Meaning
mode	`poll`, `sub` or `cyc`
req / **act**	The requested period and the period actually achieved (or the rate the server revised it to), in ms
min / **max**	The shortest and longest period seen
p50 / **p95** / **p99**	Jitter at the 50th, 95th and 99th percentiles
rtt	Read round-trip time
lat	Source-to-receive latency
age	How old the source value was when received
total	Samples acquired
rep	Repeated samples: readings whose value equalled the previous one. A controller that has frozen a register scores well on every other column; this count keeps climbing
gaps / **dup** / **ooo**	Sequence gaps, duplicate packets and out-of-order packets (for streams and subscriptions)
drop	Samples dropped
qd	Driver queue depth
ovr	Poll overruns: cycles that could not start on time

A second table gives the temporal quality of each group — where its timestamps come from (**origin**: request interval, device, server, stream arrival or client receive), latency median and spread, the timestamp uncertainty, and how many cross-group frames were coherent, degraded or incoherent. **Save report…** writes both tables to a text file with a timestamp, for the commissioning file.

The *Diagnostics and support* chapter covers this window's second table in detail. For a finished recording, **Recording ▸ Recording Timing Analysis…** gives the same evidence after the fact, and **File ▸ Export Timing Evidence (CSV)…** exports it.

Reading it: the packaging line

With the packaging line running, each machine group should show `poll`, **req** `500`, **act** close to `500`, and **ovr** `0`; the `Conveyors` and `CasePacking` groups show `200`, and `Line` `1000`. An **act** well above **req**, or a growing **ovr**, means the machine cannot read the group as fast as asked: lengthen the poll rate, split a large group, or move the connection to a less loaded machine.

Worked example: adding a tag to the packaging line

The checkweigher's conveyor belt speed is to be trended beside the case weight. On the demonstration line it is simulated; on a real line the same steps apply with a real address.

1. Open the packaging line (**Help ▸ Demonstration…**, **Packaging line (simulated)**, **Open**).
2. **Acquisition ▸ I/O Configuration…**.
3. In the I/O tree, expand **Packaging** and select the **Checkweigher** group. Check its properties: **Poll rate (ms)** `500`, **Description** *Weighs every case and signals the reject pusher.*
4. Press **+ Tag**. A new tag appears under **Checkweigher**.
5. In the tag's properties enter:

Field	Value
Name	`BeltSpeed`
Data type	`Double`
Acquisition group	`Checkweigher`
Unit	`m/min`
Scaling	factor `1`, offset `0`
Tag enabled	ticked
Wave	`Sine`
Base value	`32`
Amplitude	`1.5`
Period (s)	`120`
Noise	`0.2`

On a real Siemens checkweigher the tag would instead have an **Address** such as `DB20.DBD8` and **Data type** `Float`, and you would press **Read this tag** to confirm it reads about 32.

6. Press **OK**. The configuration validates, and a running connection takes the new tags without reconnecting; one whose connection settings you changed restarts. On a client, a new revision is activated on the node.
7. In the Signal Tree, `Packaging ▸ Checkweigher ▸ BeltSpeed` appears with its unit `m/min`. Press `F5` if acquisition is stopped.
8. Double-click `BeltSpeed` and `CaseWeight` to plot both. Because they are in the same group, each `BeltSpeed` sample has exactly the timestamp of the `CaseWeight` sample read with it.
9. **File ▸ Save Project** (`Ctrl+S`) keeps the change in the project file.

The new signal's topic is `Packaging/Checkweigher/BeltSpeed`. That is the name to use in an alarm rule, a virtual signal (`[Packaging/Checkweigher/BeltSpeed]`), a dataset or, with the Web HMI module, an operator page.

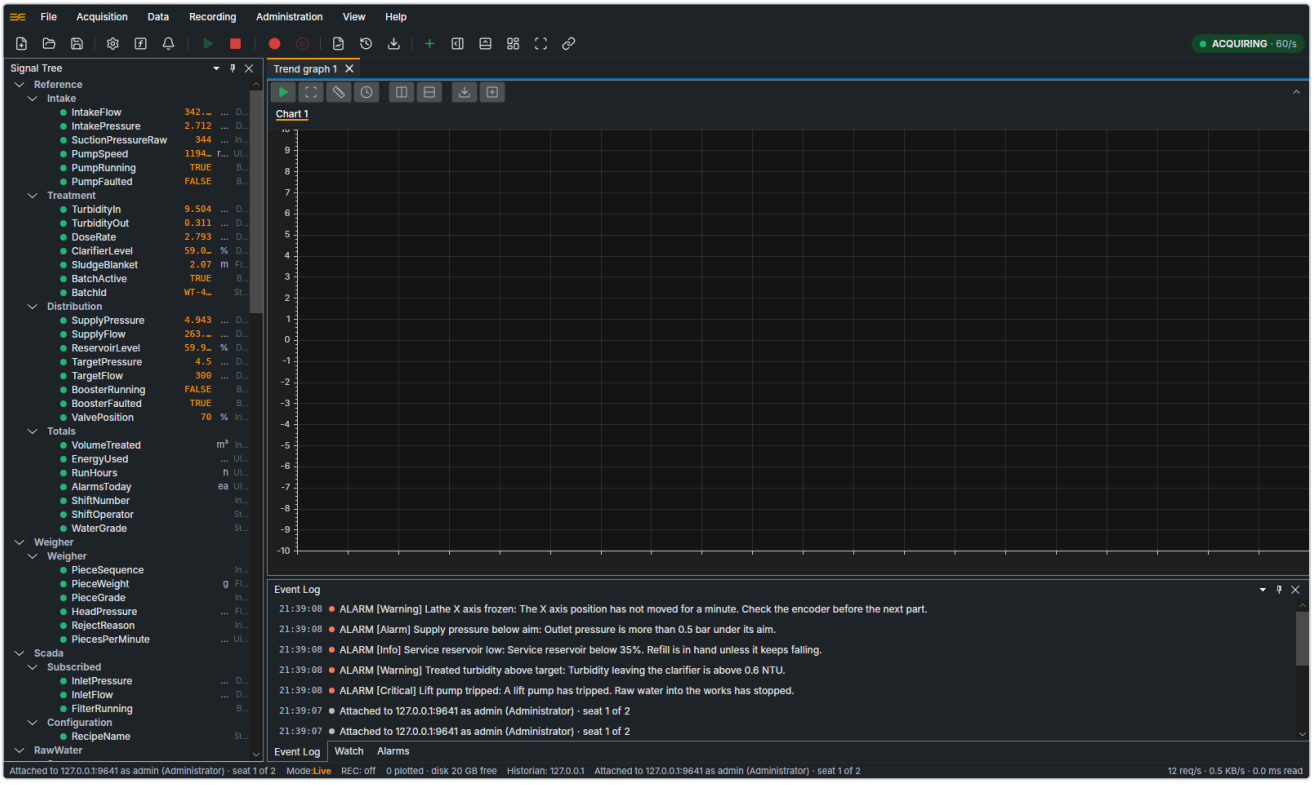
Signals: the signal tree, virtual and computed signals

A *signal* is anything Meelmo can trend, record, alarm on and store: every tag a connection acquires, and every virtual signal computed from them. This chapter describes the Signal Tree, where the signals of the open project are listed with their live values; engineering units; virtual (computed) signals and the expression language they are written in; and how digital and text signals behave.

Defining virtual signals needs the **Engineer** or **Administrator** role. Looking at the Signal Tree needs no more than **Viewer**.

The Signal Tree

The Signal Tree is the panel on the left of the main window. It mirrors the I/O configuration: connection, then acquisition group, then tag, with computed signals in their own branch.



The Signal Tree of the packaging line while acquiring: the Packaging connection, its Erector, BottomSealer, Loader, Checkweigher, TopSealer and Labeler groups, and under each the tags with a green quality dot, their unit and data type

Each row shows, left to right:

Element	Meaning
Quality dot	Green for good quality, red for bad (the connection is lost, or the driver disowned the reading)
Name	The connection, group or tag name. Groups are drawn in a heavier, muted style
Plotted badge	A small number beside a tag plotted on a trend graph: the number of the graph it is on. The tag's name is then drawn in bold in the accent colour
Live value	The latest value while acquiring

Element	Meaning
Unit	The engineering unit from the tag's configuration (`g` , `cases/min` , `%` , `°C`)
Data type	The tag's type, abbreviated (`D…` for Double, `B…` for Boolean, `In…` for Int32)

Hover over a name to see it in full when the panel is too narrow.

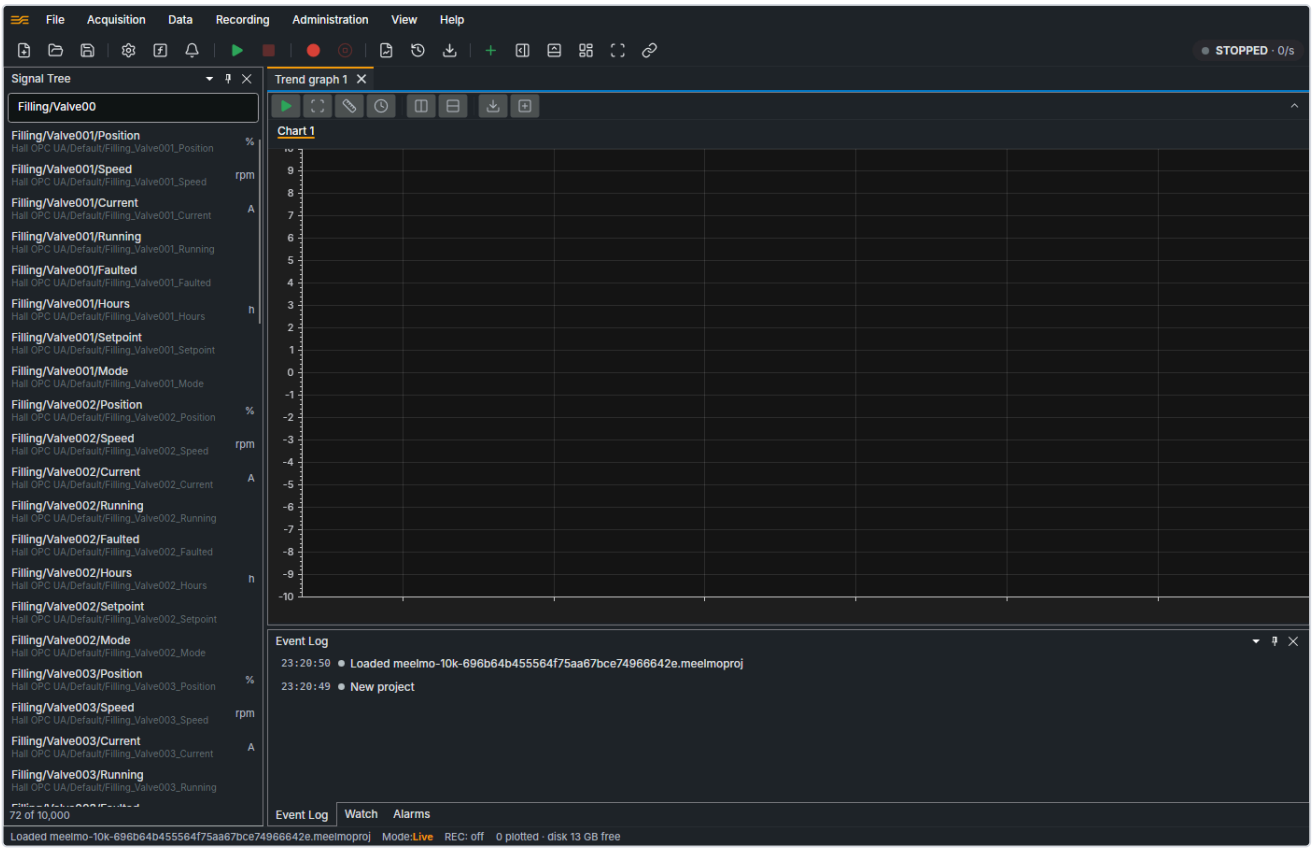
How the tree is built

- **Connections and groups are always open**, because they are the shape of the acquisition.
- **Below the group, a tag's own name is opened into branches** on `.` , `/` and the array index, because in every symbolic controller that name is a path: `"MyDB".Axis.Speed` becomes `"MyDB"` ▸ `Axis` ▸ `Speed` , and `Data[0]` … `Data[199]` become one `Data` branch. Text inside double quotes is never split, so a block called `"My.DB"` stays one level.
- **Branches with twelve children or fewer start open**; larger ones start closed, which is what makes a long array readable.
- **Computed signals** are listed under a branch **Virtual ▸ Computed**, one entry per enabled virtual signal.
- **Ungrouped tags** are listed under a group named for their poll rate (for example `r250ms`).
- **In playback**, the tree lists the signals in the open recording rather than the project's configuration.
- **A branch draws its first 250 rows**, then a row saying how many more there are — *… 9,750 more signals: search above to find them*. A tree is laid out branch inside branch and cannot draw only the rows on screen, so a group of ten thousand signals used to hold the shell up for tens of seconds; the search box above the tree finds every one of them.

Searching the tree

Type in the box above the tree. A fifth of a second after you stop typing, the tree gives way to a list of the signals that match — each with its whole name, its path, its live value and its unit — and **N of M** under it. Double-click a match to plot it, as in the tree; clear the box to get the tree back. **Enter** searches without waiting, and **Down** moves into the list.

A signal matches when its name, display name, address in the controller or description contains the text, in any case, or when its unit is the text exactly — `bar` finds signals in bar and not in mbar. This is the same search, word for word, as every list of signals on the Desktop: the tag list of the Virtual Signals window, the history browser, the tag-drift review, a tag-list import's preview, the Add Tags preview, the I/O window's tag table and Browse Tags, and every box that asks for one signal — an alarm condition, a detection or advisory rule, a dataset column, a model's target, a timing analysis's topic, a glossary entry, a buffer field's tag. A box that asks for one signal proposes the matches as you type and shows **N of M** under it; it still takes a name it does not know.



The Signal Tree searched for Filling/Valve00 in a project of 10,000 signals: 72 of 10,000 matches, each with its name, its path and its unit

When the project has no signals, the tree shows **No signals yet** — *Signals come from the controllers this project reads. Describe a connection, give it groups and tags, and they appear here — then drag one onto a trend graph.* — and a **Set up I/O…** button that opens the I/O configuration.

Working with the tree

To	Do
Plot a tag on the active trend graph	Double-click it
Remove a plotted tag from its graph	Double-click it again
Plot a tag on a particular graph or strip	Drag it onto that graph
Write a value to a writable tag	Right-click ▸ **Write value…** (see *Acquisition and I/O configuration*, *Writes*)
Start an alarm rule for a signal	Right-click ▸ **New alarm from this signal…**, which opens the alarm editor for that signal (see *Alarms*). Needs the permission to configure alarms
Add, edit, switch off or delete a connection or a tag	Right-click it, see *Configuring I/O from the tree* below
Hide or show the tree	**View** ▸ **Signal Tree**, or the panel's own title-bar controls

Configuring I/O from the tree

The tree's right-click menu also configures I/O, without going through **Acquisition ▸ I/O Configuration…** first. After **Write value…** and **New alarm from this signal…**, which are always first, it offers what fits the row you right-clicked:

Right-click on	I/O entries
The empty space below the rows	**Add connection…**, **Open I/O configuration…**
A connection	**Add tags…**, **Browse tags…**, **Edit connection…**, **Enable connection** or **Disable connection**, **Delete connection…**, **Open I/O configuration…**
A tag	**Edit tag…**, **Delete tag…**, **Open I/O configuration…**
A group, a computed signal, or a signal of a recording being played back	**Open I/O configuration…**

- **Add connection…** opens the I/O Configuration window on a new connection, with its name and its driver picker showing.
- **Edit connection…** and **Edit tag…** open the I/O Configuration window with that connection or tag selected and its branch open.
- **Add tags…** opens the **Add Tags** dialog (pattern, pasted list, Excel workbook or script) for that connection. **Browse tags…** reads the connection's tag table and offers it in the tag browser, as **Browse Tags…** does in the window. The browse's progress is shown on the status line.
- **Enable connection** and **Disable connection** switch the connection on or off, like its **Connection enabled** box.
- **Delete connection…** and **Delete tag…** ask first, in the window's words (*Remove tag 'LineRate'?*). When an alarm, a computed signal, a flow, a screen or a dataset reads it, the question lists them, the same list the window shows (see *Acquisition and I/O configuration ▸ Before a destructive change is applied*).

A change made here is the same change the window makes. It is checked by the same validation and, where something downstream depends on it, asked about with the same **What this change reaches** question. It is applied the way **OK** applies the window: on a node, as a new configuration revision activated as you and recorded with your name. Under **Restart only what needs it** the connection you changed takes the change without reconnecting. The status line then says what the node did, for example *The node applied revision 42: 0 connections restarted, 1 updated in place, 5 untouched*. This works the same on a Desktop that holds its node and on one joined to a node elsewhere.

The editing entries need the permission to configure I/O (the **Engineer** or **Administrator** role). Without it the menu shows only **Open I/O configuration…**, greyed out, and its tooltip says why. The node checks the permission again when the change is applied, whatever the menu showed.



The signal tree beside a trend graph of the packaging line: each plotted tag (CaseWeight, WeightTarget, the checkweigher Rate, SealTemperature and others) is shown in bold with the badge 1, and its unit in the right-hand column

Signal names

Every signal has a topic, `Connection/Group/Tag`, and that is how the rest of the product refers to it:

Signal	Topic
Case weight at the checkweigher	`Packaging/Checkweigher/CaseWeight`
Loader running	`Packaging/Loader/Running`
Compressed air pressure	`Packaging/Line/AirPressure`
Computed weight deviation	`Virtual/Computed/WeightDeviation`
Computed OEE	`Virtual/Computed/OEE`

Tag names repeat across groups on most real plants — every machine on the packaging line has a `Rate`, a `Running` and a `Faulted` — so the topic is the only name that is unique. Use it in expressions, alarm rules, datasets and reports.

Display names

The topic is an identity, and it reads like one: a legend of five machines' rates says `Rate` five times. A signal can also have a **display name** — what people see it called. Set it in the **Display name** field of the tag (**Acquisition ▸ I/O Configuration…**) or of the virtual signal (**Acquisition ▸ Virtual Signals…**). The packaging line's signals have them: `Packaging/Loader/Rate` is *Pouch loader rate*, `Packaging/Conveyors/BufferLoaderToCheckweigher` is *Buffer after pouch loader*, `Virtual/Computed/LoaderThroughput` is *Pouch loader throughput*.

Where	What it shows
Web client and HMI: trend legends, value grids and lists, key-value tables, deviation panels, a card's parts, the detail view	The display name, unless the page's author gave that reading a label of its own
The node's signal catalogue (REST, SDKs, OPC UA publication)	The display name, as the signal's `displayName`
Expressions, alarm rules, datasets, reports, bindings	The topic, always

A display name is one string, in the project's own language. Wording that belongs to one page, or that must change with the viewer's language, is still a label on that page's binding — a key into the HMI project's string table (see *The web HMI Studio*).

Leave it empty and nothing changes: the signal is called by its name, which is how every project saved before display names existed reads. Changing or clearing a display name never changes a topic, so no binding, rule or history is affected.

Units

A signal's engineering unit is set where the signal is defined:

- for a tag, in the **Unit** field of the tag in **Acquisition ▸ I/O Configuration…**;
- for a virtual signal, in the **Unit** field of **Acquisition ▸ Virtual Signals…**.

The unit is shown in the Signal Tree, in a trend's legend and axis, in the Watch Window, in exported files, in datasets and reports, and is stored with the history. A unit is a label: Meelmo does not convert between units. To show a value in another unit, either scale the tag (in the I/O configuration, engineering value = raw × factor + offset — for example a transmitter delivering 0–27648 counts for 0–10 bar has factor `0.000361690` and offset `0`) or define a virtual signal that converts it (`[SimPLC/Fast/Temperature] * 1.8 + 32`, unit `°F`).

Use the same spelling for the same unit everywhere (`cases/min`, not sometimes `cpm`): a report or a dataset that groups signals by unit sees two spellings as two units.

Virtual signals

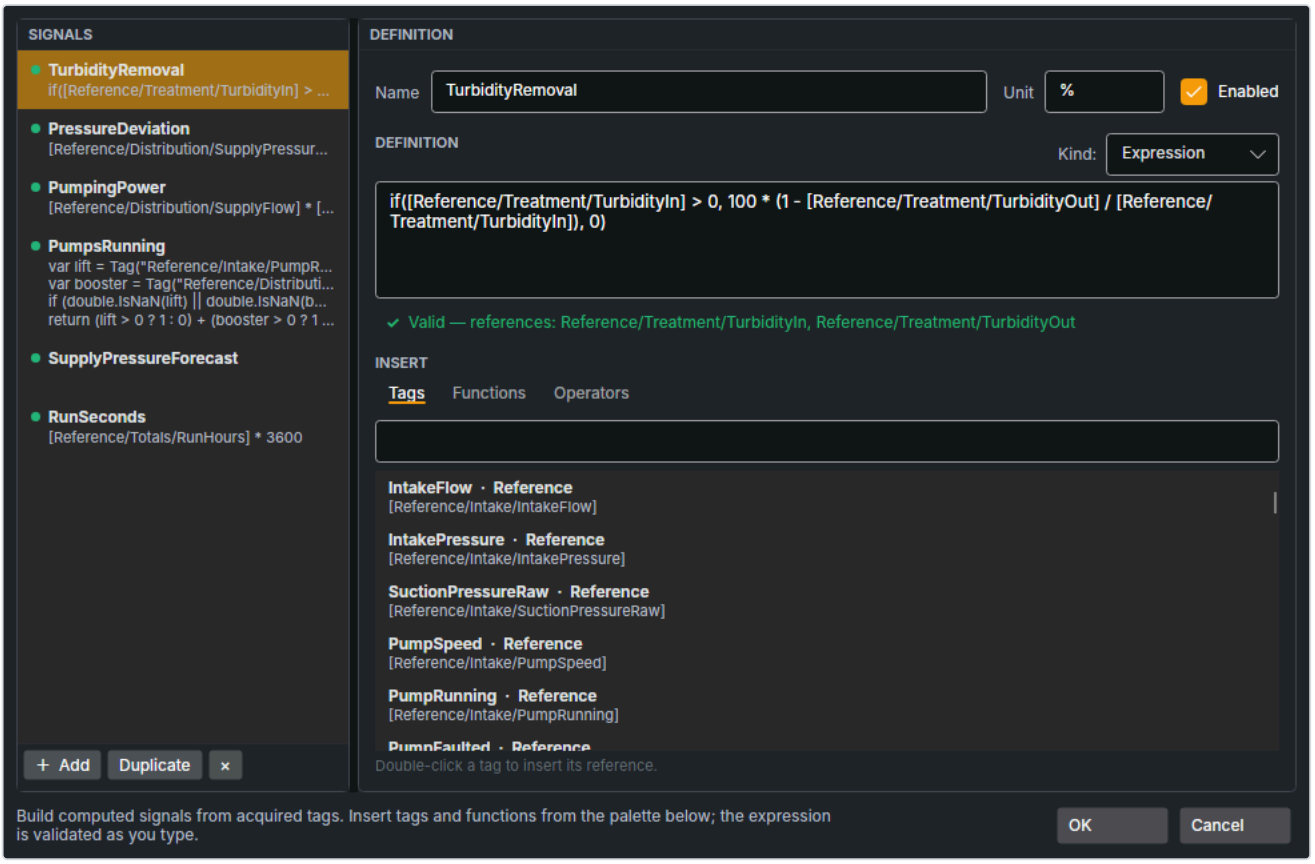
A **virtual signal** is a channel computed from acquired signals by an expression, evaluated live as the inputs arrive. It is acquired, recorded, historized, alarmed and plotted exactly like a measured one. Its topic is `Virtual/Computed/<Name>`.

Typical uses:

- **Derived process values**: a deviation from target, an efficiency, a mass flow from a volume flow and a density.
- **Unit conversion**: °C to °F, counts to engineering units.
- **State codes**: collapsing a machine's `Running`, `Faulted`, `Blocked` and `Starved` bits into one number an operator page or a report can colour.
- **KPIs**: availability, performance, quality and OEE.
- **Time-aware values**: a moving average, a rate of change, a transport delay between two gauges on a continuous line.

The Virtual Signals window

Open it with **Acquisition ▸ Virtual Signals…** or the virtual signals button on the toolbar.



The Virtual Signals window for the packaging line: the SIGNALS list on the left shows ErectorState, BottomSealerState, LoaderState and the other per-machine state and throughput signals; the DEFINITION pane shows ErectorState's nested conditional expression, marked Valid, with the Tags tab of the INSERT palette listing the Packaging tags

The SIGNALS list (left) shows every virtual signal with its name and the start of its expression, and a dot that is green while its definition is valid. Below it:

Button	What it does
+ Add	Adds a new, empty virtual signal
Duplicate	Copies the selected signal, as a starting point for a similar one
×	Removes the selected signal

The DEFINITION pane (right) edits the selected signal:

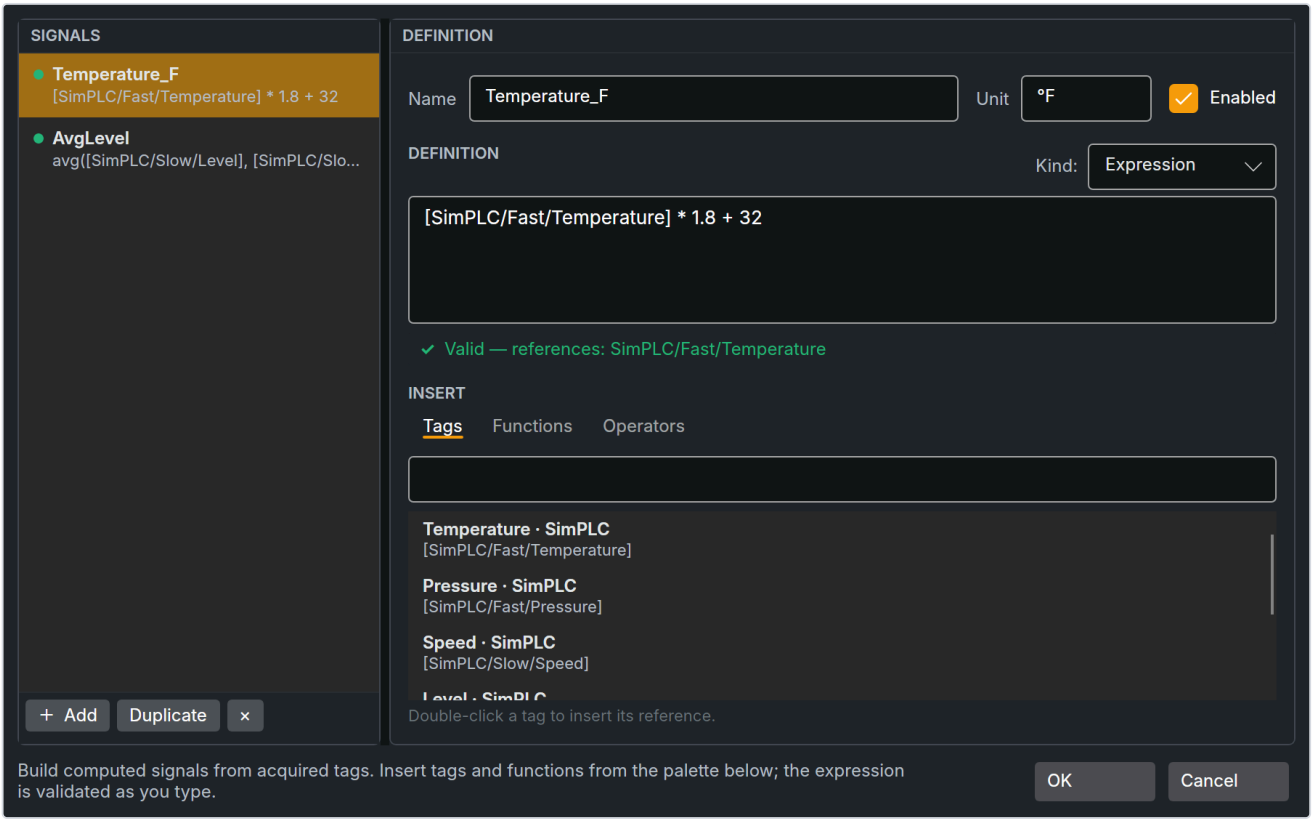
Field	Meaning
Name	The signal's name, the last part of its topic `Virtual/Computed/<Name>`. Must be unique among the virtual signals
Unit	Its engineering unit
Display name	What screens call it; empty is its name. See *Display names* above
Enabled	Cleared, the signal is kept but not computed and not shown in the Signal Tree
Kind	**Expression** (the built-in expression language), **C# Script**, or **Model** — an ONNX model from the node's registry, fed features of other signals. Model signals belong to the Models module and are described in its chapter.
Definition box	The expression itself

Field	Meaning
Status line	Validated as you type: ✓ *Valid — references: …* listing every signal the expression reads, or the first error and where it is

The INSERT palette (below the definition) inserts text at the cursor:

Tab	Contents
Tags	Every acquired tag, with its connection and topic. Type in the search box to filter; *Double-click a tag to insert its reference.*
Functions	Every function, grouped as Math, Rounding, Select and Trig, with a description. *Double-click a function to insert it.*
Operators	Buttons for the operators, comparisons, logic, the conditional, and the constants `pi`, `e`, `true` and `false`

OK validates every signal and applies them (acquisition picks up the new definitions); **Cancel** discards the changes. The window's footer summarises: *Build computed signals from acquired tags. Insert tags and functions from the palette below; the expression is validated as you type.*



*The Virtual Signals window from an earlier build with a small test project: Temperature_F defined as [SimPLC/Fast/Temperature] * 1.8 + 32 with unit °F, and AvgLevel as an avg() of two levels*

Referencing signals

Inside an expression, an acquired tag is referenced:

- by topic in square brackets: `[Packaging/Checkweigher/CaseWeight]` — always unambiguous, and what the **Tags** palette inserts;
- or by the tag's bare name: `CaseWeight` — convenient on a small project, but ambiguous where names repeat across groups, as they do on the packaging line.

A virtual signal is computed from acquired tags. To reuse part of another virtual signal's formula, repeat that part in the new expression: the packaging line's `OEE` repeats the formulas of `Availability`, `Performance` and `Quality` rather than referring to them.

The expression language

Element	Syntax
Numbers	`12`, `0.5`, `1.2e-3`
Constants	`pi`, `e`, `true` (1), `false` (0)
Arithmetic	`+ - * / %` (remainder) and `^` (power), with parentheses
Unary	`-x`, `+x`, `!x` (logical not)
Comparisons	`< <= > >= == !=`, each giving 1 or 0
Logic	`&&`, `` ` ``
Conditional	`cond ? a : b`, or `if(cond, a, b)`

Functions:

Group	Functions
Math	`abs(x)`, `sqrt(x)`, `pow(x, y)`, `exp(x)`, `log(x [, base])`, `log10(x)`, `sign(x)`, `mod(a, b)` (floored), `hypot(a, b)`
Rounding	`floor(x)`, `ceil(x)`, `round(x [, digits])`, `trunc(x)`, `frac(x)`
Select	`min(a, b, …)`, `max(a, b, …)`, `avg(a, b, …)`, `clamp(x, lo, hi)`, `if(cond, a, b)`
Trigonometry	`sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `atan2(y, x)`, `sinh`, `cosh`, `tanh` (radians), `deg(x)`, `rad(x)`

Functions that read the past

These functions keep a short history of their input, so they answer questions about time:

Function	Answers
`delay(x, s)`	What `x` was `s` seconds ago — a transport delay
`movavg(x, s)`	The mean of `x` over the last `s` seconds
`tmax(x, s)` / `tmin(x, s)`	The highest and lowest value of `x` over the last `s` seconds
`tstd(x, s)`	The standard deviation of `x` over the last `s` seconds
`deriv(x)`	The rate of change of `x`, per second
`integ(x)`	The running integral of `x`, in units × seconds
`edge(x)`	1 on the reading where `x` became true, 0 otherwise

Points to know:

- **Each call site keeps its own history**, so `delay(a, 2) - delay(b, 4)` is two independent questions. No call site keeps more than an hour.

- **They are blank until enough history has accumulated**, rather than falling back to the oldest reading: `movavg(x, 300)` gives no value for its first five minutes after acquisition starts. A delay that quietly returned the wrong instant would be worse than one that says it does not know yet.
- `delay` **interpolates** between the two readings either side of the moment asked for.
- **They move with the clock.** A signal using them is re-evaluated even when its inputs have not changed, so an integral of a steady value keeps climbing.
- The status line cannot show a value for them while you edit, because there is no history yet.

`delay` is what a line with conveyors between its machines needs. The bottom sealer and the top sealer work on the same case only after the travel time between them, so `[Packaging/TopSealer/SealTemperature] - delay([Packaging/BottomSealer/SealTemperature], 12)` is a comparison, and subtracting the two directly is not.

When an input is missing

A reference that cannot be resolved — a misspelt topic, or a tag that has not delivered a value yet — yields "not a number", which propagates through the expression. The virtual signal is then marked **bad quality** for that sample rather than producing a confident wrong answer. A division by zero is handled the same way; guard against it explicitly where zero is a normal value, as the packaging line does with `+ 0.001` in its denominators, or with `if()`:

```
if([Reference/Treatment/TurbidityIn] > 0, 100 * (1 - [Reference/Treatment/TurbidityOut] /
[Reference/Treatment/TurbidityIn]), 0)
```

C# script signals

For the rare case the expression language cannot express, set **Kind** to **C# Script**. The definition is C# that returns a `double`; `Tag("name")` reads a live value by tag name or topic and returns `NaN` where there is none, and `Math` is imported. A **Validate** button appears: press it to compile the script (*Script compiles.* or the compiler's error). The reference works' `PumpsRunning` is an example:

```
var lift = Tag("Reference/Intake/PumpRunning");
var booster = Tag("Reference/Distribution/BoosterRunning");
if (double.IsNaN(lift) || double.IsNaN(booster)) return 0;
return (lift > 0 ? 1 : 0) + (booster > 0 ? 1 : 0);
```

Prefer an expression wherever one will do: it is validated as you type, it can use the functions that read the past, and anybody can read it.

The packaging line's computed signals

The packaging line defines 28 virtual signals. They are worth reading as examples:

Signal	Unit	Expression (abridged)	What it gives
`ErectorState` (one per machine)	—	`[…/Faulted] > 0 ? 3 : […/Blocked] > 0 ? 2 : […/Starved] > 0 ? 1 : […/Running] > 0 ? 0 : -1`	A state code: 3 faulted, 2 blocked, 1 starved, 0 running, −1 stopped
`ErectorThroughput` (one per machine)	cases/min	`[…/Running] > 0 && […/Faulted] < 1 && […/Blocked] < 1 && […/Starved] < 1 ? […/Rate] : 0`	The machine's rate while it is genuinely producing

Signal	Unit	Expression (abridged)	What it gives
`Availability`	%	`movavg([Packaging/Loader/Running], 300) * 100`	Share of the last five minutes the loader, which sets the line rate, was running
`Performance`	%	`min(100, [Packaging/Loader/Rate] / ([Packaging/Line/TargetRate] + 0.001) * 100)`	Rate against target
`Quality`	%	`GoodCount / (GoodCount + RejectCount + 0.001) * 100`	Good cases as a share of all cases
`OEE`	%	Availability × Performance × Quality	Overall equipment effectiveness
`WeightDeviation`	g	`[Packaging/Checkweigher/CaseWeight] - [Packaging/Checkweigher/WeightTarget]`	Case weight against specification
`UnaccountedUnits`	cases	`InfeedCount - GoodCount - RejectCount`	Cases that went in and neither came out nor were rejected
`OrderProgress`	%	`min(100, GoodCount / (OrderTarget + 0.001) * 100)`	Progress through the production order

The alarm rule *Case weight out of tolerance* is written over one of them: `abs([Virtual/Computed/WeightDeviation]) > 15`.

Worked example: a computed signal

The quality engineer wants to see whether the case weight is drifting, without the case-to-case scatter: a one-minute moving average of the weight deviation, and a flag that says when the average has left a ±5 g band.

1. Open the packaging line and choose **Acquisition ▸ Virtual Signals…**.
2. Select `WeightDeviation` in the list and press **Duplicate**. A copy is added and selected.
3. Change **Name** to `WeightDrift`, keep **Unit** `g`, keep **Enabled** ticked and **Kind Expression**.
4. Replace the definition with:

   ```
   movavg([Packaging/Checkweigher/CaseWeight] - [Packaging/Checkweigher/WeightTarget], 60)
   ```

   To build it from the palette instead: clear the box, open **Functions**, double-click `movavg(x, s)`; open **Tags**, type `CaseWeight` in the search box and double-click `[Packaging/Checkweigher/CaseWeight]`; open **Operators** and press **−**; double-click `[Packaging/Checkweigher/WeightTarget]`; type `, 60)`.
5. The status line reads *✓ Valid — references: Packaging/Checkweigher/CaseWeight, Packaging/Checkweigher/WeightTarget*.
6. Press **+ Add**, set **Name** to `WeightDriftOut`, leave **Unit** empty, and enter:

   ```
   abs(movavg([Packaging/Checkweigher/CaseWeight] - [Packaging/Checkweigher/WeightTarget], 60))
   > 5
   ```

It is 1 while the one-minute average is more than 5 g away from target and 0 otherwise. The drift formula is repeated rather than referring to `WeightDrift`, because a virtual signal is computed from acquired tags.

7. Press **OK**. Both appear in the Signal Tree under **Virtual ▸ Computed**.
8. If acquisition is not running, press `F5`. Double-click `WeightDeviation`, `WeightDrift` and `WeightDriftOut` to plot them. For the first minute `WeightDrift` is blank; then it follows `WeightDeviation` as a smooth line, and `WeightDriftOut` draws as a 0/1 state band.
9. **File ▸ Save Project** to keep them.

`Virtual/Computed/WeightDrift` can now be recorded and historized with the rest, used in a dataset, or alarmed on — for example a rule `[Virtual/Computed/WeightDriftOut] > 0` with a raise delay of 30 s, as described in the *Alarms* chapter.

Digital and text signals

Not every signal is a number on a scale.

Boolean signals (`Running`, `Faulted`, `ChangeoverActive`) are drawn on a trend as filled state bands on their own 0/1 axis, in the order they were added: a run of TRUE is a solid block, so a sequence reads as a timing diagram rather than as a square wave. Because 0 and 1 are real values, a Boolean can be used in expressions (`[Packaging/Loader/Running] > 0`), averaged (`movavg` of a Boolean is the fraction of time it was true, which is how `Availability` is built), integrated (accumulated on-time in seconds) and analysed like any other signal. A comparison in an expression produces a 0/1 signal of the same kind, such as `WeightDriftOut` above.

Integer signals (`Int16` … `UInt64`, such as `Loader/PouchesPerCase` or `CasePacking/PalletCount`) behave as numbers.

Text signals (`String` tags such as `Line/OrderBatch`, `Line/ProductCode`, `CasePacking/CaseState` and `CasePacking/RobotState`) are drawn as a categorical text track: coloured, labelled segments, one per run of the same value, with the current value shown in the legend as `state (RUN)`. A text track has no number behind it, so it has no Y scale, no reference line and no analysis, and is not offered as one half of an X-Y plot. For the same reason a text signal cannot be used inside an expression; a virtual signal is always numeric.

Text signals are recorded and historized like any other, and are what a dataset or a detection rule uses to tell one product or order from the next.

The trend graphs themselves — strips, axes, cursors and analysis — are described in the *Trends* chapter.

Trend graphs

A trend graph is where you look at signals over time in the Desktop: live while the plant is being acquired, or from a recording during playback. This chapter explains how to put signals on a graph, how to arrange axes and strips, how booleans and text signals are drawn, how to zoom and go back, how to measure with cursors, how to leave notes that are saved with the project, what the per-channel analysis tools do, how to work with several graphs at once, and how to export what you see.

Everything in this chapter works the same way live and in playback. Where the two differ (zooming during playback re-reads the recording, for example), the difference is stated.

What a trend graph is for

A trend graph answers questions of the form "what did these signals do, and when, relative to each other?". Typical uses on a packaging line:

- watching `Packaging/Checkweigher/CaseWeight` against its `WeightTarget` while a new product is run in;
- lining up `Packaging/Loader/Running`, `Faulted`, `Blocked` and `Starved` to see which state a station spent the last hour in;
- comparing the `Rate` of every station to find which one is holding the line back;
- measuring the time between a fault and the moment the station downstream starved.

A trend graph draws what acquisition delivers (or what a recording contains). It does not create data: if nothing is being acquired and no recording is open, the graph is empty. Stored history over weeks and months is read with **Data ▸ Browse History…**, which is described in *The historian* chapter; its results plot on the same kind of graph.

Rendering follows the size of the chart on the screen rather than the number of samples behind it, so a trace holding a million samples draws about as quickly as one holding a thousand.

Putting signals on a graph

When a project is open, the **Signal Tree** panel on the left lists every connection, acquisition group and tag, and below them the virtual signals (for the packaging demonstration, the `Packaging` connection with groups such as `Erector`, `Loader` and `Checkweigher`, then `Virtual ▸ Computed` with `OEE`, `Quality`, `WeightDeviation` and the other computed signals).

To plot a signal:

1. Start acquisition with **Acquisition ▸ Start** (`F5`) or the green start button on the toolbar, or open a recording.
2. Drag a tag from the **Signal Tree** onto a trend graph. Dragging onto an empty part of the document area creates a new graph for it. Double-clicking a tag plots it, and double-clicking a plotted tag takes it off again.
3. Drag more signals onto the same graph to compare them.

A signal that is plotted is shown in **bold** in the tree, with a small badge giving the number of the trend graph (or graphs) it is on, for example `1` or `2,3`. That badge is how you find where a signal lives when several graphs are open.



A trend graph of the packaging line split into strips: CaseWeight against its WeightTarget, the checkweigher rate, the bottom sealer's seal temperature, air pressure and electrical power over ten minutes. Each legend shows the latest value and the visible range, and the plotted tags carry the badge 1 in the tree.

To add an empty graph, use **View ▸ Add Trend Graph**, or the green **+** button on the main toolbar.

Worked example: the packaging demonstration's graphs

The demonstration project `examples/demo/packaging.meelmoproj` is saved with three trend graphs, and opening it restores them:

Graph	Where it docks	What it shows
Process	fills the document area	three strips: the Erector's `MagazineLevel`, `Rate` and `BlanksPerMinute`; the BottomSealer's `Rate`, `SealPressure` and `SealTemperature`; and the Loader's `PouchInfeedRate`, `Rate`, `CollatorBacklog` and `PouchesPerCase`
States	below	the Erector's `Running`, `Faulted`, `Blocked` and `Starved`, and the BottomSealer's `Running` and `Faulted`, as digital state bars
Computed	to the right	one strip each for `Availability`, `OEE`, `OrderProgress` and `Performance` from `Virtual/Computed`

In the Process graph the Loader strip puts `CollatorBacklog` and `PouchesPerCase` on one shared axis group called `ea` (both are counts of items), while the rates keep their own axes. That arrangement is described under *Axis layouts and strips* below.

The parts of a trend graph



Three trend graphs tiled in the document area. Each has its own tab, its chart toolbar, a legend line per trace with the live value and the visible minimum and maximum, and value bars down the right-hand edge.

Each trend graph is a docked document with a tab title (**Trend graph 1**, **Trend graph 2** …) and, inside it, one or more chart tabs (**Chart 1** …). The parts are:

Part	What it shows or does
Graph tab	The graph's title. Drag it to re-dock the graph; the × closes it.
Chart toolbar	The row of icon buttons described below. The chevron at its right end collapses it.
Chart tabs	Several charts inside one graph, each with its own signals. Right-click a tab for **Rename** (asks for a **Tab name**) and to close it.
Legend	One entry per trace: the colour, the signal name, the current value in brackets, and the minimum … maximum over the visible window in square brackets. The × removes the trace from the chart.
Plot area	The traces, the time axis along the bottom, and one or more value axes.
Value bars	A bar per trace at the right-hand edge, showing the trace's current reading against its own range, so a glance tells you where in its span each signal is.

The chart toolbar

The chart toolbar is icons only; each button has a tooltip.

Button	Tooltip	What it does
Play (green triangle)	**Resume live — jump to now and follow new data**	After a pan has taken the chart away from the present, returns to now and follows new data. While the chart is behind, the tooltip reads *Following … behind — press to catch up to now*. During playback it runs the recording.
Fit (corners)	**Zoom to fit all data**	Shows everything the chart holds and discards the zoom history.

Button	Tooltip	What it does
Ruler	**A/B measurement: click two points to read Δt, Δ and slope**	Turns on the measurement cursors (see *Cursors and measurement*).
Clock	**Show/hide the time navigator (overview strip) for this chart**	An overview strip of the whole time range, for moving around a long recording.
Side-by-side axes	**Stack Y axes horizontally: each analog signal its own side-by-side axis**	Gives every analog trace its own Y axis.
Stacked bands	**Stack signals vertically: one band per signal with its own aligned axis**	Splits the chart into one strip per signal on a shared time axis.
Download arrow	**Save the active chart as an image**	Saves a picture of the chart.
Plus in a square	**New chart tab**	Adds a chart tab inside this graph.

To hide the chart toolbar on every graph at once (for a wall display, for example), use **View ▸ Chart toolbar** or `Ctrl+Shift+T`, or right-click the toolbar strip, or click the chevron at its right end. All four do the same thing, to every graph. The setting is remembered between sessions. **View ▸ Toolbar** does the same for the main toolbar, and **View ▸ Full screen** (`F11`) removes the window chrome; `Esc` always leaves full screen.

The chart's right-click menu

Right-clicking the plot area opens the chart's context menu. It is built from the signals on the chart, so its first entries are one submenu per trace (named after the signal, with its colour), followed by items for the chart as a whole. The per-trace submenus are described in *Per-trace appearance* and *Per-channel analysis*; the chart items are:

Item	What it does
Split channels into strips	One strip per signal (see *Strips*).
Stack by value axis	One strip per axis group, so signals that share an axis share a strip.
Merge into one strip	Back to a single strip holding every signal.
Reset band heights	Offered once you have dragged a strip boundary; returns every strip to an equal share.
Add empty strip	Adds an empty strip to drop signals into.
Clear channels in this strip	Removes every signal from the strip you clicked in.
Remove this strip	Offered when there is more than one strip.
Data table (visible window)…	Lists the raw samples behind what is drawn.

Item	What it does
Clear detected events	Offered after **Detect threshold crossings…**; removes the markers.
Annotations	**Add note here…**, and near an existing note **Edit note…** and **Remove note**; **Remove all notes** clears them.
Time format	`HH:mm:ss`, `HH:mm:ss.fff`, `yyyy-MM-dd HH:mm:ss` or `yyyy-MM-dd` on the time axis.
Show legend / **Legend position**	Shows or hides the legend; places it **Top-right**, **Top-left**, **Bottom-right** or **Bottom-left**.
Show grid	Grid lines on or off.
Measurement mode (A/B)	The same as the toolbar's ruler.
Zoom keeps following live	See *Zooming a live chart*.
Zoom back / **Zoom forward**	Steps through the zoom history.
Zoom to fit	Shows all data and clears the zoom history.
Clear all channels	Removes every signal from the chart.
Process context	Offered only where the node has a semantic model. Shades the chart with the states the model recorded over the visible window (for example running, changeover, fault); **None** removes the shading. While the node is being asked it reads *Asking the model…*, and *The model holds nothing for this window* when there is nothing to show.
Save image…	Saves the chart as a picture.

Per-trace appearance

Right-click the plot and open the submenu named after a trace to change how that one trace is drawn. For an analog signal the submenu offers:

Item	Choices
Shared Y axis / **Own axis (left)** / **Own axis (right)**	Which Y axis the trace uses (see below).
Axis group	Put the trace on a named axis shared with other traces; **New axis group…** asks for a **Group name**.
Show value axis	Show or hide this trace's scale, to give the width back to the plot.
Line / **Markers** / **Line + markers**	How samples are drawn.
Filled area (to zero)	Fills between the trace and zero (not offered for booleans).
Colour	**Auto (palette)** takes the colour of the trace's position; a swatch sets a fixed colour; **Custom…** opens a colour picker.
Line width	`1 px` to `4 px`.
Line style	**Solid**, **Dashed**, **Dotted**.
Y axis scale	**Auto**, or **Set range…** for fixed limits.
Reference line	**Add…** draws a horizontal limit line at a value you type; **Clear** removes them.

Item	Choices
Analyse	The analysis tools (see *Per-channel analysis*).
Move to new strip	Moves the trace into a strip of its own.
Remove channel	Takes it off the chart.

Colours are assigned by position: the first trace on a chart takes the first palette colour, the second the second, and so on. That keeps screenshots consistent between graphs. A colour you choose explicitly sticks to that signal.

Worked example: a weight chart with limits

To watch case weights against the tolerance used by the demonstration's alarm rule (`abs(WeightDeviation) > 15`, see the *Alarms* chapter):

1. Drag `Packaging/Checkweigher/CaseWeight` and `Packaging/Checkweigher/WeightTarget` onto a new trend graph. Both are in grams, around `1240`.
2. Right-click the plot, open **CaseWeight ▸ Axis group ▸ New axis group…**, and type `g`. Do the same for **WeightTarget**, choosing the existing group `g`. The two now share one scale, so their difference is visible.
3. Open **CaseWeight ▸ Reference line ▸ Add…** and type `1255`; add a second reference line at `1225`. These are the ±15 g limits.
4. Open **CaseWeight ▸ Y axis scale ▸ Set range…** and set `1200` to `1280`, so that the chart does not rescale as the weight moves and two runs can be compared by eye.
5. Save the project (`Ctrl+S`); the arrangement is saved with it.

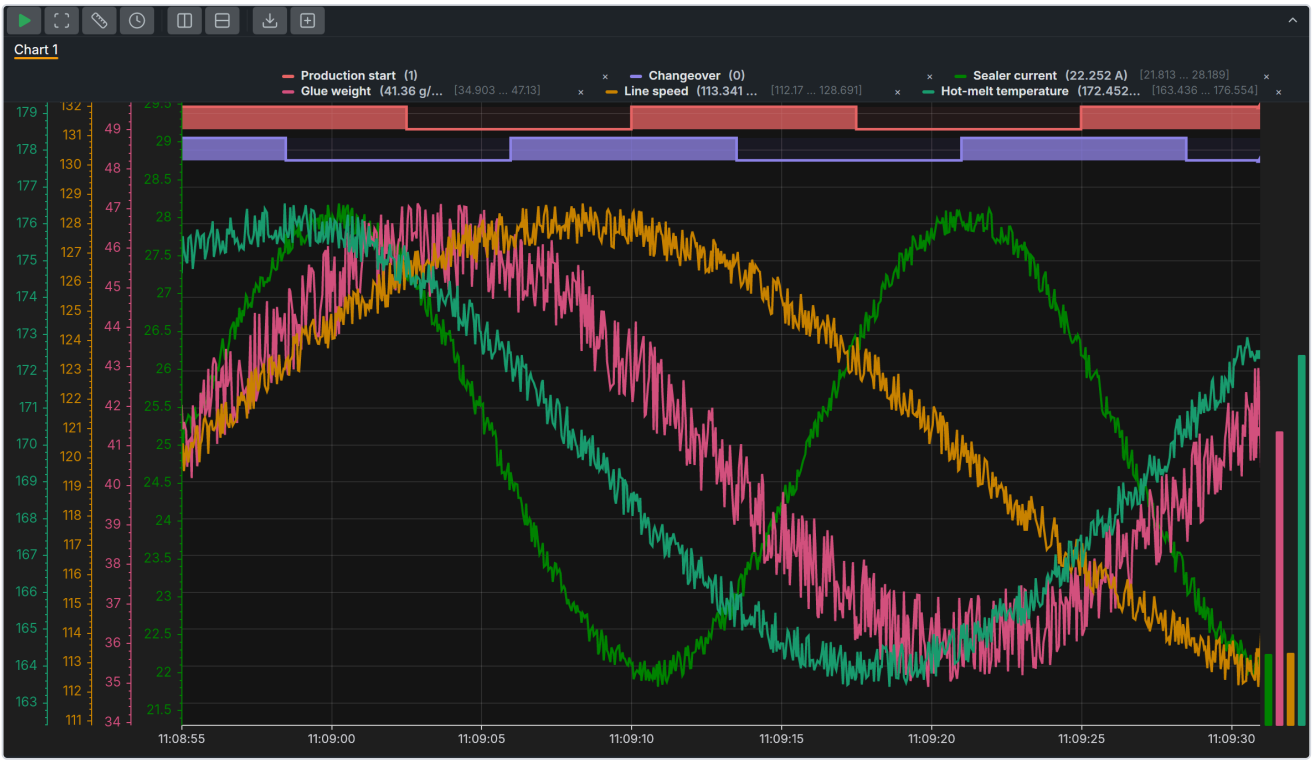
Axis layouts and strips

There are two independent ways to arrange traces, and they combine.

Y axes side by side

With **Shared Y axis** every trace is drawn against one scale. That is right for signals with the same unit and range (the ten station `Rate` signals, all in `cases/min` around 30 to 34), and wrong for signals of different sizes, where the small one becomes a flat line.

Own axis (left) and **Own axis (right)** give a trace its own scale at the left or right edge, tinted in the trace's colour. The toolbar's side-by-side axes button does this for every analog trace at once. An **axis group** sits between the two: several traces on one named scale, with other traces on theirs.



A chart with four analog traces, each on its own Y axis tinted in the trace's colour, above two digital state bars. The legend shows each trace's value and visible range, and value bars run down the right-hand edge.

Strips

A strip is a horizontal band of the chart with its own Y axes; all strips share the time axis, so a moment lines up vertically across them. Strips are the better choice when signals are unrelated in size and you want each to use the full height of its band.

- **Split channels into strips** (or the toolbar's stacked-bands button) gives each signal its own strip.
- **Stack by value axis** gives each axis group its own strip.
- **Merge into one strip** puts everything back into one.
- **Add empty strip**, **Move to new strip**, **Clear channels in this strip** and **Remove this strip** build an arrangement by hand.
- Drag the boundary between two strips to change their heights; **Reset band heights** evens them out again.



Three trend graphs with their signals split into strips, one band per signal, sharing each graph's time axis.



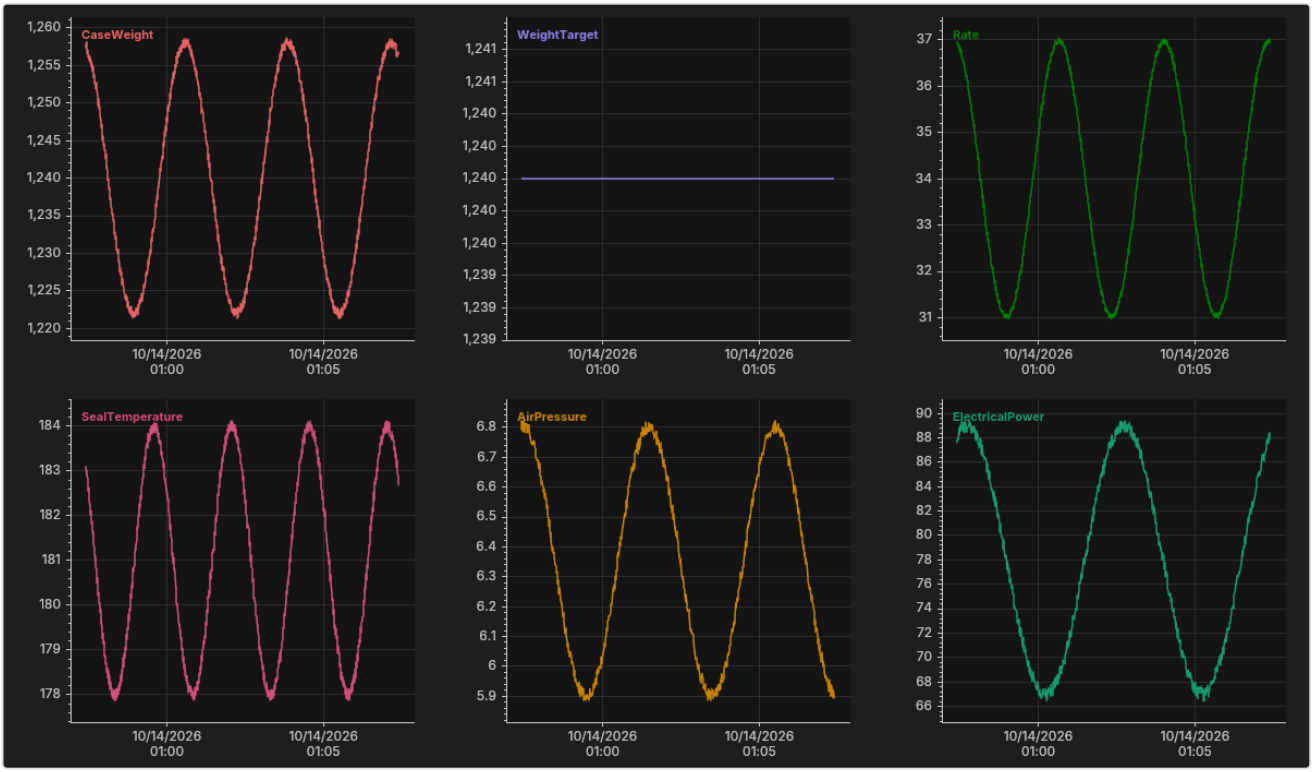
The same graphs with the Y axes of the stacked signals each drawn at their own scale, tinted per trace.

The View ▸ Panels menu

View ▸ Panels applies a layout to the active trend graph without right-clicking a trace. It needs at least one plotted signal; otherwise it says *Plot one or more signals first — panels give each plotted signal its own pane*.

Item	What it does
Merge into one strip	All channels overlaid in one pane.
Stacked (shared time)	One pane per channel, applied to the graph itself: same channels, same clock, same cursors and value bars. Switching back is **Merge into one strip**.
Grid	Opens a separate window, **Grid of panels**, with one pane per channel (see below).

The Grid of panels window. Each plotted analog or boolean signal gets a small pane of its own, laid out in a near-square grid: four signals make two columns of two, five or six make three columns of two, ten make four columns of three. Each pane carries the signal's name in its own colour in the top-left corner, its own value scale on the left and its own dated time axis along the bottom, so every pane reads on its own. Text signals are left out, because a categorical track has no line to draw. The window holds what was in view on the graph at the moment it opened: it is a snapshot for comparing shapes side by side, and it does not follow the plant or a later zoom. Close it and choose **Grid** again for a fresh one. It opens beside the main window, and you can keep several open at once.



The Grid of panels window for six signals of the packaging line in three columns of two: CaseWeight, WeightTarget, Rate, SealTemperature, AirPressure and ElectricalPower, each with its name in its own colour, its own value scale and a dated time axis

Use **Stacked (shared time)** when you want each signal in its own band and still live; use **Grid** when you want to compare the shapes of many signals at a glance, for example the ten station `Rate` signals of the packaging line, each at its own scale.

The same **Panels** submenu is at the end of each trace's **Analyse** submenu.

The layout of every graph (its strips, axis modes, axis groups, colours and chart tabs) is saved with the project, and with an analysis preset (see *Exporting and analysis presets*).

Digital and text signals

Boolean signals (for example `Packaging/Loader/Running` or `Packaging/Line/ChangeoverActive`) are drawn as filled state bars on their own 0/1 lane, in the order they were added. A run of TRUE is a solid block, so several booleans read as a timing diagram rather than as square waves to be traced by eye. Booleans keep the full per-trace submenu: 0 and 1 are real values, so a spectrum of a digital finds its cycle rate, and its integral is the accumulated on-time. A boolean is not offered as either axis of an X-Y plot.

Text signals (for example `Packaging/Line/ProductCode`, `Packaging/Line/OrderBatch` or `Packaging/CasePacking/CaseState`) are drawn as a categorical track: coloured, labelled segments, one per run of the same value, with the current value in the legend. When `ProductCode` changes from `PCH-241-R` to `PCH-318-G` a new segment starts, which is the quickest way to see which product was running when something happened.

A text track has nothing numeric behind it, so its submenu is short: **Colour**, **Analyse ▸ Panels**, **Move to new strip** and **Remove channel**. There is no Y scale, reference line or numeric analysis, and a text signal is not offered as the other half of an X-Y plot.

Worked example: which state was the Loader in?

1. Add a trend graph and drag `Packaging/Loader/Running`, `Faulted`, `Blocked` and `Starved` onto it. Four state bars appear.
2. Drag `Packaging/Line/ProductCode` onto the same graph. A text track shows which product was running.
3. Drag `Virtual/Computed/LoaderState` onto it as well. This virtual signal is `3` while faulted, `2` blocked, `1` starved, `0` running and `-1` stopped, so it gives one analog line that summarises the four bars.
4. Right-click and choose **Split channels into strips** so each lane has its own height.

Zooming, panning and going back

- **Drag** sideways to pan; **scroll** the mouse wheel to zoom the time axis.
- During playback, a zoom re-reads a finer slice of the recording rather than magnifying what is already drawn, so zooming in reveals individual samples.
- **Zoom to fit** (toolbar, context menu, **View ▸ Zoom to Fit**, or the main toolbar's fit button) shows all the data the chart holds.

Zoom back and forward

Investigating an event is usually a sequence of zooms: into the hour the fault happened, then the minute, then the two seconds around the step. **Zoom back** and **Zoom forward** on the chart's context menu step through that sequence, up to 64 steps deep, the way a web browser's back and forward buttons do. **Zoom to fit** starts a new sequence.

Zooming a live chart

On a live chart, panning and zooming mean different things:

- **Dragging sideways keeps the width** of the window and asks about a *different* time, so the chart stops following the plant and stays where you took it. The Play button then shows how far behind now you are; press it to catch up.
- **Scrolling changes the width** and asks about the *same* time more closely, so the chart keeps following with the right edge pinned to now.

Turn **Zoom keeps following live** off on the context menu if you want any gesture to stop the chart following. A live zoom is limited to between one second and 24 hours.

Two settings on the **View** menu affect how live charts draw:

Setting	Effect
Scroll live charts continuously	The window follows the clock instead of waiting for a reading, so the trend slides smoothly between samples. It costs a repaint of every live chart 25 times a second; on a busy chart that can mean dropped frames.
Draw live traces from kept columns	Analog traces are drawn from a decimation kept as readings arrive instead of walking every sample on every frame (measured at 3.2 ms per frame against 60.2 ms). It stands aside where it would be meaningless: sparse data, or a window you have panned or zoomed.

Cursors and measurement

Turn on **Measurement mode (A/B)** from the context menu, or press the ruler button on the chart toolbar. A crosshair follows the mouse and snaps to samples, reading every trace at that instant. Click to drop cursor **A**, then click again to drop cursor **B**. A measurement grid opens below the chart with one row per trace:

Column	Meaning
Name	The trace.
X1, **X2**	The times of cursors A and B.
X2-X1	The time between them.
Y1, **Y2**	The trace's value at A and at B.
Y2-Y1	The change between them.



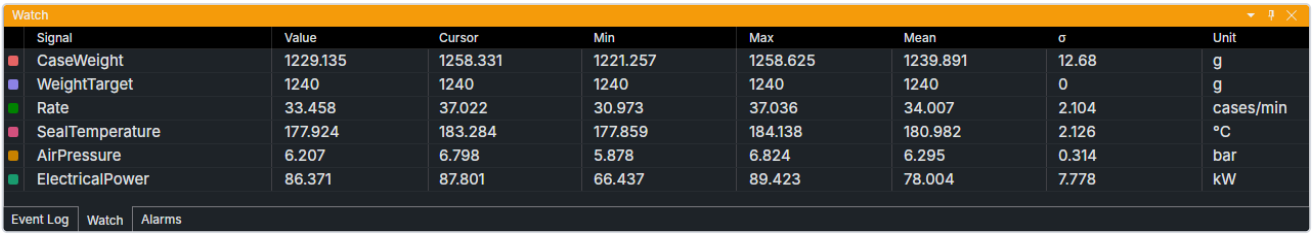
Trend graph 1 with measurement cursors A and B dropped eight seconds apart; the grid below reads X1, X2, X2-X1, Y1, Y2 and Y2-Y1 for each trace. Trend graph 2 shows state bars and analog strips, and trend graph 3 two traces on separate axes.

Worked example: how long until the line starved?

1. Plot `Packaging/Loader/Faulted` and `Packaging/Checkweigher/Starved` on one graph.
2. Turn on measurement mode.
3. Drop **A** on the rising edge of `Faulted` and **B** on the rising edge of `Starved`.
4. **X2-X1** is the time the buffer conveyor between them (`Packaging/Conveyors/BufferLoaderToCheckweigher`) held the Checkweigher fed after the Loader stopped.

Statistics: the Watch panel

The **Watch** panel (**View ▸ Watch Window**) lists every signal plotted on the active graph with statistics over the visible time window, refreshed once a second. Its tab in the bottom dock is captioned **Watch**.



The Watch panel for six signals of the packaging line over a ten-minute window: Value, Cursor, Min, Max, Mean, σ and Unit for each, with the trace colour beside the name

Column	Meaning
(colour)	The trace's colour on the chart.
Signal	The signal's short name.

Column	Meaning
Value	Its latest value.
Cursor	Its value under the measurement cursor, when measurement mode is on.
Min, **Max**, **Mean**	Over the visible window.
σ	Standard deviation over the visible window.
Unit	The tag's engineering unit.

Because the statistics are over the visible window, zooming changes them: zoom to the last ten minutes and **Mean** is the ten-minute mean. The legend's square brackets show the same visible minimum and maximum.

Notes that are saved with the project

Everything else on a chart (the cursors, detected crossings, overlays) is temporary and gone when the project is next opened. A note is not.

1. Right-click at the moment you want to mark and choose **Annotations ▸ Add note here…**.
2. Type the text, for example `Film reel changed, WO-84121`, and confirm.
3. The note is drawn as a labelled vertical line behind the traces.
4. Save the project. The note is stored with the chart tab it was written on, with the name of the person who wrote it.

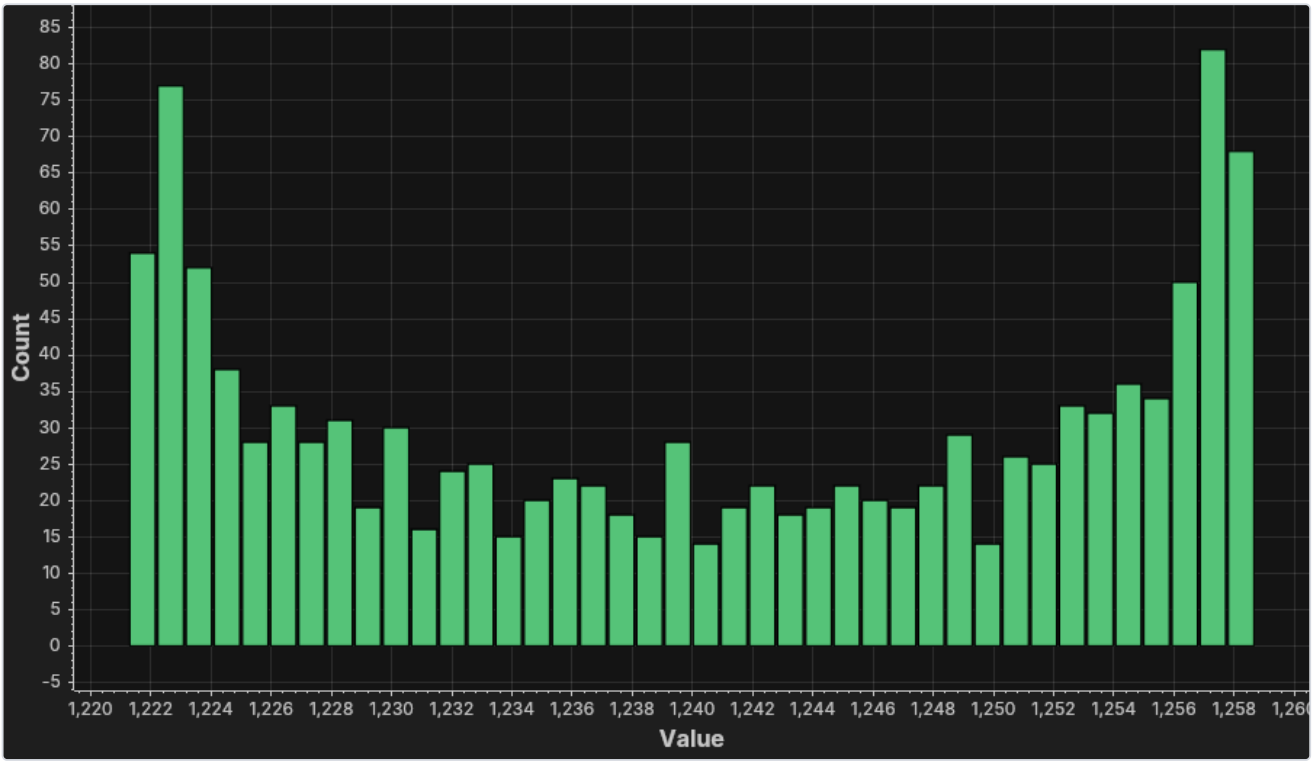
Right-clicking near a note offers **Edit note…** and **Remove note**; **Remove all notes** clears every note on the chart. A note is stored as an instant with its time-zone offset, so it points at the same moment when the project is opened somewhere else.

Per-channel analysis

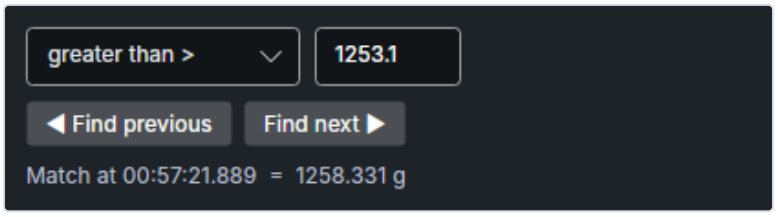
Right-click the plot and open **(signal name) ▸ Analyse** for the tools that work on one trace. They operate on the data the chart holds for that trace.

Tool	What it does
Frequency spectrum (FFT)	Opens the trace's spectrum, for finding a cycle rate, a beat or a resonance.
Histogram	Opens the distribution of the trace's readings.
X-Y plot against	Plots this trace against another plotted analog signal, resampled onto a common time base, with the Pearson correlation *r*. Offered only for analog traces and only when another analog trace is plotted.
Overlay moving average…	Adds a smoothed companion trace; you choose the window.
Overlay low-pass filter…	Adds a filtered companion trace.
Overlay RMS (window)…	Adds a rolling RMS trace.
Overlay envelope (min/max)…	Adds the rolling minimum and maximum.

Tool	What it does
Overlay derivative (d/dt)	Adds the rate of change.
Overlay integral (∫)	Adds the accumulation (for a boolean, the on-time).
Find… (search & jump to a condition)	Searches the trace for a condition and moves the view to it.
Detect threshold crossings…	Marks every crossing of a value you type; **Clear detected events** removes the marks.
Live scope	Opens a live oscilloscope-style sweep of this signal's last 1,000 readings in its own window (analog only; see *The live scope and the live logger*).
Live logger	Opens a live logger that keeps every reading of this signal since it opened, in its own window (analog only).
Panels	**Stacked (shared time)** or **Grid**, as on the View menu.



Histogram — CaseWeight: the distribution of the case weight's readings over ten minutes of the packaging line, between about 1,221 and 1,259 g



Find — CaseWeight: the condition greater than 1258.3, Find previous and Find next, and the match the view moved to

The live scope and the live logger

A trend graph is a window onto a clock. Two kinds of signal do not fit that window well: a fast signal whose waveform you want to see cycle by cycle, and a slow ramp you want to watch whole, from the moment you start looking. For these, **Analyse ▸ Live scope** and **Analyse ▸ Live logger** open the trace in a window of its own. Both are offered for analog signals only (not for booleans or text), and both open beside the main window without changing the graph you opened them from; you can keep several open at once.

	Live scope	**Live logger**
Window title	**Scope — *signal***	**Logger — *signal***
What it draws	The signal's most recent 1,000 readings, against their position rather than their time	Every reading since the window opened, against time
How it moves	**Sweep**: like an oscilloscope, the trace is redrawn from left to right over the old one, with a small blank gap running ahead of the newest reading	**Accumulate**: the trace grows to the right and the time axis widens to show all of it
Value axis	Rescales continuously to the readings on screen	Follows the readings as they arrive
Capacity	1,000 readings; older ones are overwritten	Grows for as long as the window is open
Use it for	A cycle, a beat or a ringing that a time window blurs	A cold start, a warm-up or a fill, watched from its start

Both windows take the same readings the trend graph receives from live acquisition, so they draw only while acquisition is running and show nothing during playback of a recording. Closing the window stops it listening; nothing is saved. The scope and logger windows follow the light or dark theme.

Worked example: watching the seal temperature recover after a stop

1. Start the packaging demonstration (`F5`) and plot `Packaging/BottomSealer/SealTemperature`.
2. Right-click the plot and choose **SealTemperature ▸ Analyse ▸ Live logger**. The **Logger — SealTemperature** window opens and starts drawing from now.
3. Leave it open across a stop and restart of the sealer. Because the logger keeps every reading since it opened, the whole dip and recovery stays on screen however long it takes, while the trend graph's window moves on.
4. For the shape of each cycle instead, choose **SealTemperature ▸ Analyse ▸ Live scope**: the last 1,000 readings are swept across the window, so a periodic disturbance shows up as the same shape drawn over itself.

The X-Y plot is what to use for a characteristic curve, which cannot be shown on a time axis at all. The two signals are paired on the X signal's timestamps within the visible window, so the curve describes the window you are looking at.

Data table (visible window)… on the chart menu lists the raw samples behind what is drawn. Use it to settle whether the chart is smoothing something.

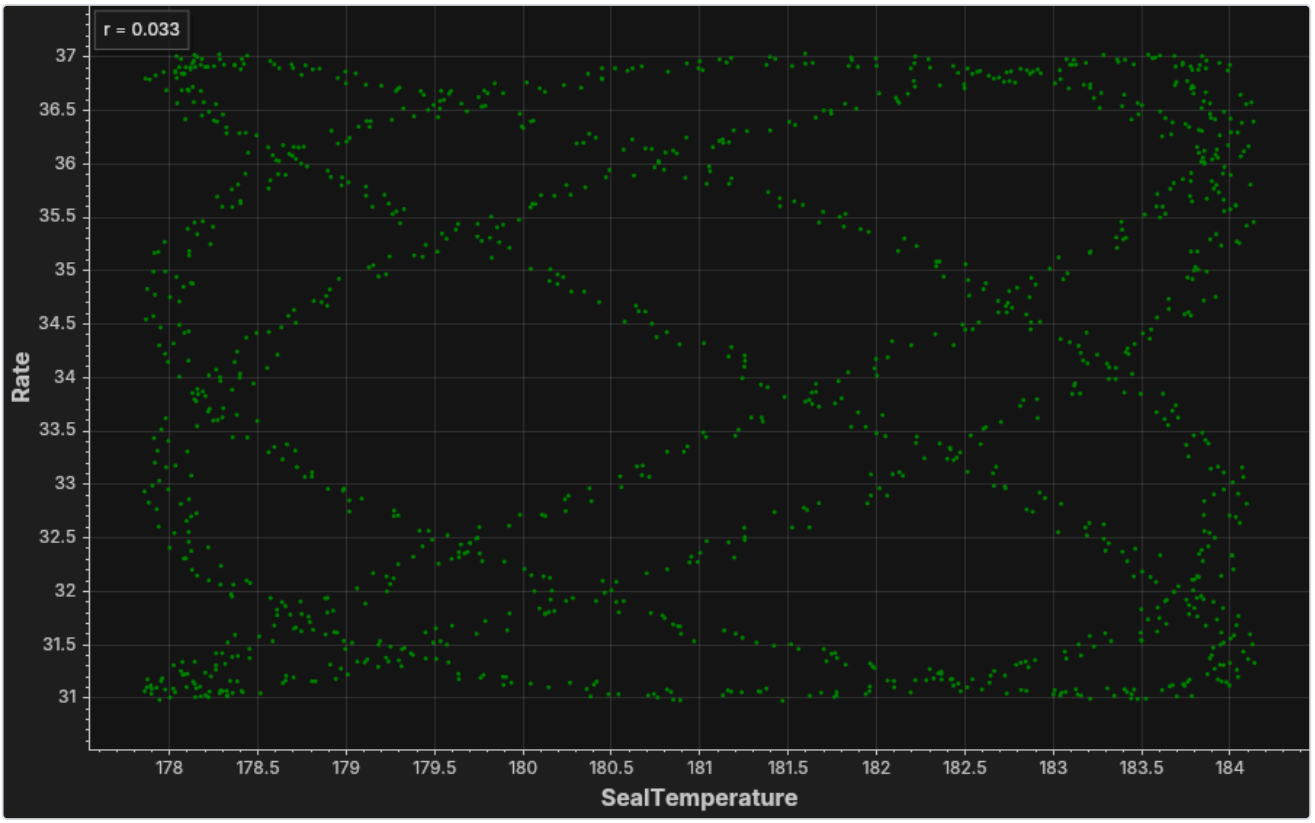
Time	CaseWeight	WeightTarget	Rate	SealTemperature	AirPressure	ElectricalPower
00:57:15.889	1257.656 g	1240 g	36.943 cases/min	183.939 °C	6.767 bar	87.297 kW
00:57:16.396	1258.612 g	1240 g	37.009 cases/min	183.842 °C	6.789 bar	87.143 kW
00:57:16.889	1258.421 g	1240 g	36.97 cases/min	183.866 °C	6.789 bar	87.143 kW
00:57:17.389	1258.592 g	1240 g	36.91 cases/min	183.692 °C	6.757 bar	87.253 kW
00:57:17.889	1258.527 g	1240 g	36.911 cases/min	183.777 °C	6.757 bar	87.253 kW
00:57:18.389	1257.294 g	1240 g	36.945 cases/min	183.732 °C	6.803 bar	87.848 kW
00:57:18.889	1258.394 g	1240 g	36.995 cases/min	183.576 °C	6.803 bar	87.848 kW
00:57:19.389	1258.073 g	1240 g	36.989 cases/min	183.561 °C	6.77 bar	87.846 kW
00:57:19.889	1257.54 g	1240 g	37.022 cases/min	183.537 °C	6.77 bar	87.846 kW
00:57:20.389	1257.519 g	1240 g	37.01 cases/min	183.685 °C	6.787 bar	87.338 kW
00:57:20.889	1257.944 g	1240 g	36.975 cases/min	183.361 °C	6.787 bar	87.338 kW
00:57:21.389	1258.698 g	1240 g	37.014 cases/min	183.603 °C	6.798 bar	87.801 kW
00:57:21.889	1258.331 g	1240 g	37.022 cases/min	183.284 °C	6.798 bar	87.801 kW
00:57:22.389	1257.624 g	1240 g	36.955 cases/min	183.368 °C	6.779 bar	87.238 kW
00:57:22.889	1258.609 g	1240 g	36.931 cases/min	183.455 °C	6.779 bar	87.238 kW
00:57:23.389	1257.886 g	1240 g	36.975 cases/min	183.406 °C	6.802 bar	87.435 kW
00:57:23.889	1258.17 g	1240 g	36.947 cases/min	183.185 °C	6.802 bar	87.435 kW
00:57:24.408	1257.23 g	1240 g	36.93 cases/min	183.129 °C	6.8 bar	88.553 kW
00:57:24.889	1258.285 g	1240 g	36.926 cases/min	183.107 °C	6.8 bar	88.553 kW
00:57:25.389	1257.3 g	1240 g	36.979 cases/min	183.074 °C	6.792 bar	87.564 kW
00:57:25.889	1257.236 g	1240 g	36.974 cases/min	183.116 °C	6.792 bar	87.564 kW
00:57:26.389	1257.738 g	1240 g	36.931 cases/min	183.094 °C	6.821 bar	87.833 kW

1200 rows · 6 signals

Data table — visible window: one row per sample time with a column for each of the six plotted signals in its unit, and the count of rows and signals at the foot

Worked example: does seal temperature follow line speed?

1. Plot `Packaging/BottomSealer/SealTemperature` (°C, around 178) and `Packaging/BottomSealer/Rate` (cases/min).
2. Zoom to a window of at least a few minutes.
3. Right-click, open **SealTemperature ▸ Analyse ▸ X-Y plot against ▸ Rate**.
4. The window shows temperature against rate with *r*. A value of *r* near zero means the seal temperature is not tracking speed in this window.



X-Y — Rate vs SealTemperature: the top sealer's seal temperature against the checkweigher's rate over ten minutes, r = 0.033 in the corner, so the two are not tracking each other in this window

Several graphs, docking and tiling

The document area holds as many trend graphs as you need.

Command	Where	What it does
Add Trend Graph	**View** menu, main toolbar	Adds a graph as a new tab.
New trend graph — split right	**View** menu, main toolbar	Adds a graph beside the active one.
New trend graph — split down	**View** menu, main toolbar	Adds a graph below the active one.
Clear Charts	**View** menu	Removes every signal from every graph (the graphs stay).
Sync Time Axis	**View** menu, main toolbar (a toggle that stays lit)	Locks every graph to the same time window, so panning one pans them all.

Drag a graph's tab to re-dock it elsewhere, or out of the window to float it. A floating graph has a **Dock to main** button that brings it back. The arrangement is saved with the project.

Use **Sync Time Axis** when comparing sections of the line (the Erector graph above the Palletiser graph); turn it off when one graph is a zoomed detail of another.

The other panels (**Signal Tree**, **Watch Window**, **Event Log**, **Alarms Panel**) are shown and hidden from the **View** menu. Each panel's title bar can float it, auto-hide it to a tab on the window edge (the pin), or close it; the trend graphs grow into the space.

Exporting and analysis presets

Command	What you get
File ▸ Export Plotted Signals to CSV… (`Ctrl+E` , or the main toolbar's download button)	The signals on the active graph as a CSV file (`export.csv` by default). Live, it writes the samples the graph holds; in playback it reads the plotted signals from the recording itself.
File ▸ Export Plotted Signals to Excel…	The same as an Excel workbook (`export.xlsx`).
File ▸ Export Report (HTML)…	An HTML document (`report.html`) with a picture of each strip of the active graph and a table of each signal's minimum, maximum, mean, RMS and standard deviation.
File ▸ Export Timing Evidence (CSV)…	Enabled in playback only: per-sample timing bounds from the open recording, for a commissioning file. See the *Recording and playback* chapter.
Chart toolbar download button, or context menu **Save image…**	The chart as a picture.

The export commands need at least one plotted signal; otherwise they say *Plot one or more signals first, then export.*

An **analysis preset** saves the current trend graphs, their signals and their layout to a `.opsanalysis` file, so the same view can be applied to another project or recording:

- **File ▸ Save Analysis (chart preset)…** writes the preset (it needs at least one plotted signal).
- **File ▸ Apply Analysis (chart preset)…** (also on the main toolbar) opens one and rebuilds the graphs from it.

Worked example: a shift handover pack

1. Open the Process graph of the demonstration and zoom to the last eight hours.
2. Add a note at each stop worth explaining.
3. **File ▸ Export Report (HTML)…** and save as `shift-A-2026-09-29.html` .
4. **File ▸ Export Plotted Signals to Excel…** for the numbers behind it.
5. Save the project so the notes are kept.

Keyboard and mouse summary

Gesture	Does
Drag a tag from the Signal Tree	Plot it on the graph you drop it on, or on a new graph in empty space
Drag in the plot	Pan
Mouse wheel in the plot	Zoom the time axis
Right-click in the plot	Chart and per-trace menu
`Ctrl+E`	Export plotted signals to CSV
`Ctrl+Shift+T`	Show or hide the chart toolbar on every graph
`F11` / `Esc`	Full screen on or off / leave full screen

Recording and playback

A recording is a file that captures acquired samples exactly as they arrived, so that a run can be replayed, measured and analysed later on another machine. This chapter covers starting and stopping recordings, the recording settings (continuous, rotating files and lossless capture), triggered recording, browsing recordings, the headless recorder service and how to watch it, the recording timing analysis, playback, and the `.opslog` file format.

Recording and the historian are different things. A recording is a **file** you start, stop, copy and send to somebody. The historian is a **store** with retention and rollups that answers questions over months; it is described in *The historian* chapter. Many sites use both: the historian keeps everything at a modest resolution, and recordings capture the runs worth examining in detail.

The Recording menu

Everything to do with recording is on the **Recording** menu:

Item	What it does
Start Recording (`Ctrl+R`)	Starts writing a recording. Disabled while recording.
Stop Recording (`Ctrl+Shift+R`)	Stops and closes the file. Disabled while not recording.
Continuous (rotating, record everything)	A tick: record into rotating, retention-managed files instead of one file per start/stop.
Lossless (never drop samples, back-pressure)	A tick: never drop a sample; acquisition waits for the writer if it falls behind.
Recording settings…	Where recordings go, rotation, retention, compression, automatic start.
Trigger…	Record automatically while a condition is true.
Browse Recordings…	Lists the recordings in the recordings directory and opens one for playback.
Recorder Service…	Configures and controls the headless recorder that runs as a Windows service.
Recorder Status…	Watches a headless recorder over its status API.
Recording Timing Analysis…	Reads a finished recording and reports what its timestamps are worth.

The main toolbar carries the same start (red circle) and stop buttons, and they are disabled when they cannot be used. While a recording is running, the right-hand end of the toolbar shows a red **REC** pill with the number of samples recorded, and the status bar shows `REC:` followed by the file being written (`REC: off` when nothing is being recorded).

Recording by hand

1. Start acquisition (**Acquisition ▸ Start**, `F5`).
2. Choose **Recording ▸ Start Recording** (`Ctrl+R`). The status line reads *Recording to …* with the file name.
3. Work as normal. Recording and viewing are independent: you can plot, zoom and measure while the file is being written, and what is recorded does not depend on what is plotted.
4. Choose **Recording ▸ Stop Recording** (`Ctrl+Shift+R`). The status line reads *Recording stopped*.

By default a recording is written to the node's recordings directory (under the state directory, `recordings`) with a name of the form `recording_YYYYMMDD_HHMMSS.opslog`. Change the directory in **Recording settings…**.

What a recording contains

A recording captures every acquired value, including text values, with:

- its **quality** code;
- **both** timestamps: when the value was true at the controller (source time) and when this process received it (receive time);
- a **sequence number** where the driver supplies one;
- where the timing estimator is on, the **uncertainty interval** around the source time and where that time came from.

A sample takes about 16 bytes. The file is crash-safe, describes its own signals (the signal tree is rebuilt from the file when it is opened), and can be **read while it is still being written**.

Recording settings

Recording ▸ Recording settings… opens the window titled *Where recordings go, and how long they are kept*. The settings are saved to the node (the status line confirms *Recording settings saved to the node.*), so they are part of the node's configuration and apply whichever workstation starts the recording.



The Recording settings window with the default values: the node's default recordings directory, automatic start and continuous off, rotation at 256 MB or 60 minutes, seven days' retention, compression of rotated files ticked, and lossless off.

Field	Meaning	Default
Directory	Where recordings are written. Empty means the node's default recordings directory. **Browse…** picks a folder.	empty
Start recording when acquisition starts	Records every time acquisition starts, without pressing Record.	off
Continuous: rotating, retention-managed files	Records into a sequence of files that roll over by size or age, instead of one growing file per start/stop. The same as the **Continuous** tick on the menu.	off
Rotate at (MB)	With continuous on, start a new file when the current one reaches this size. `0` means no size rotation.	`256`
Rotate after (minutes)	With continuous on, start a new file after this long. `0` means no time rotation.	`60`
Keep for (days)	With continuous on, delete rolled files older than this. `0` keeps them forever.	`7`
Compress rotated files	Compress (gzip) each file once it has rolled. The file being written stays plain so it can be read live.	on
Lossless: never drop a sample	The same as the **Lossless** tick on the menu.	off

The rotation, retention and compression fields are greyed out until continuous recording is ticked.

Lossless or best-effort

- **Best-effort** (lossless off): if the writer falls behind, the oldest queued samples are dropped so that acquisition never stalls. Every dropped sample is counted, not lost silently. The counters are Enqueued, Persisted, Evicted and Rejected, with `Dropped = Evicted + Rejected`; **Administration ▸ Diagnostics…** shows recorded and dropped totals.
- **Lossless**: if the writer queue fills, acquisition waits until it has space. Use it for a recording that has to be complete; it is slower when a disk cannot keep up, and it says so if it still cannot.

Continuous recording and editions

Continuous capture and the headless recorder service are Professional features. On an edition that does not include them, starting a continuous recording records to a single file instead and says so: *The … edition records to a single file; continuous rotating capture is not included. Recording to one file instead.*

Worked example: a week of continuous recording on the packaging line

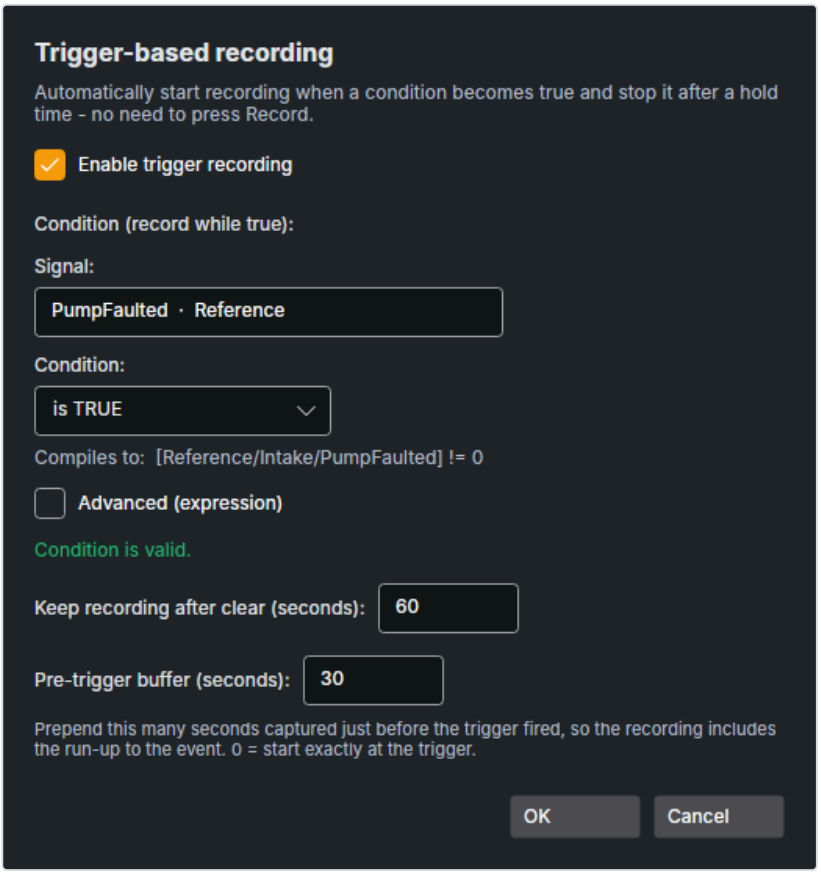
The demonstration project ships with recording set to single-file, 256 MB / 60 min rotation and 7 days' retention. To keep a rolling week of everything the line does:

1. Open **Recording ▸ Recording settings…**.
2. Tick **Start recording when acquisition starts** and **Continuous: rotating, retention-managed files**.
3. Leave **Rotate at (MB)** at `256`, set **Rotate after (minutes)** to `60`, and set **Keep for (days)** to `7`. Leave **Compress rotated files** ticked.
4. Leave **Lossless** off: the line's 13 acquisition groups poll at 200 to 1000 ms, and it is better to count a dropped sample than to delay acquisition.
5. Click **OK**, then start acquisition. The status line reads *Continuous recording to …*, and a new file begins every hour.

Triggered recording

A trigger records only while something interesting is happening, so you do not have to press Record at the right moment or search a long file for the event.

Recording ▸ Trigger… opens *Trigger-based recording*: *Automatically start recording when a condition becomes true and stop it after a hold time - no need to press Record.*



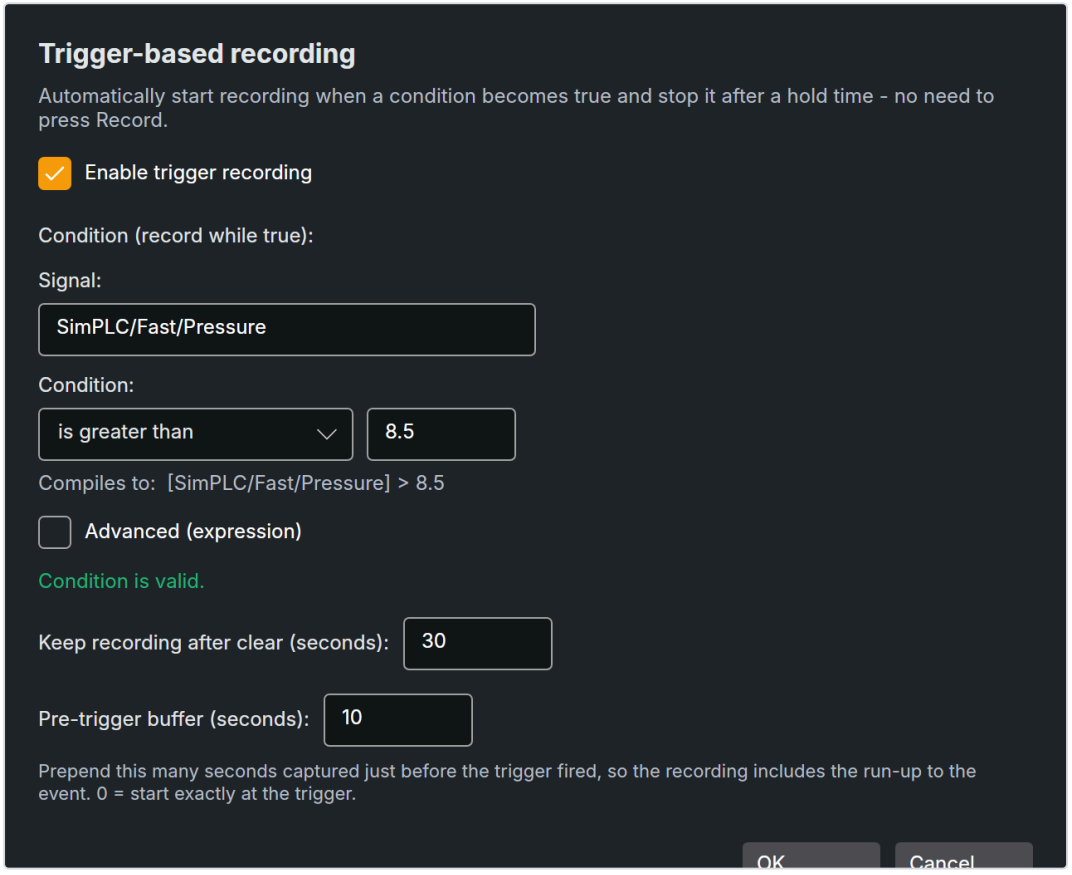
The trigger window as it opens: trigger recording not enabled, an empty signal, the comparison set to is greater than, a hold of 5 seconds after clear and no pre-trigger buffer.

Field	Meaning
Enable trigger recording	Arms the trigger. While armed the trigger controls recording, and Record and Stop are not needed.
Condition (record while true)	What to record on. Built with the three fields below, or typed as an expression.
Signal	The signal the condition tests.
Condition	The comparison: **is greater than**, **is greater than or equal to**, **is less than**, **is less than or equal to**, **equals**, **does not equal**, **is inside range**, **is outside range**; for a boolean **is TRUE** or **is FALSE**. A range comparison shows a second value box.
(value)	The value to compare with.
Compiles to	The expression the guided fields produce, for example `[Packaging/Line/AirPressure] < 6`.
Advanced (expression)	Type any boolean expression over tags instead, using `[connection/group/tag]` for a signal, for example `[Packaging/Loader/Faulted] != 0 && [Packaging/Line/ChangeoverActive] == 0`.

Field	Meaning
(validation)	*Condition is valid.*, or *Invalid:* with the reason.
Keep recording after clear (seconds)	How long the condition must stay false before recording stops, so a brief dip does not cut the file in two. Default `5`.
Pre-trigger buffer (seconds)	Seconds captured just before the trigger fired that are put at the start of the recording, so it includes the run-up to the event. `0` starts exactly at the trigger.

While armed, the status reads **TRIG armed**; while the trigger is recording it reads **TRIG recording**, and the event log records *Trigger fired - recording started.* and *Trigger cleared - recording stopped.* Stopping recording by hand while a trigger is armed disarms it (*Recording stopped; trigger disarmed.*).

The trigger is evaluated the same way in the Desktop and in the headless recorder service.



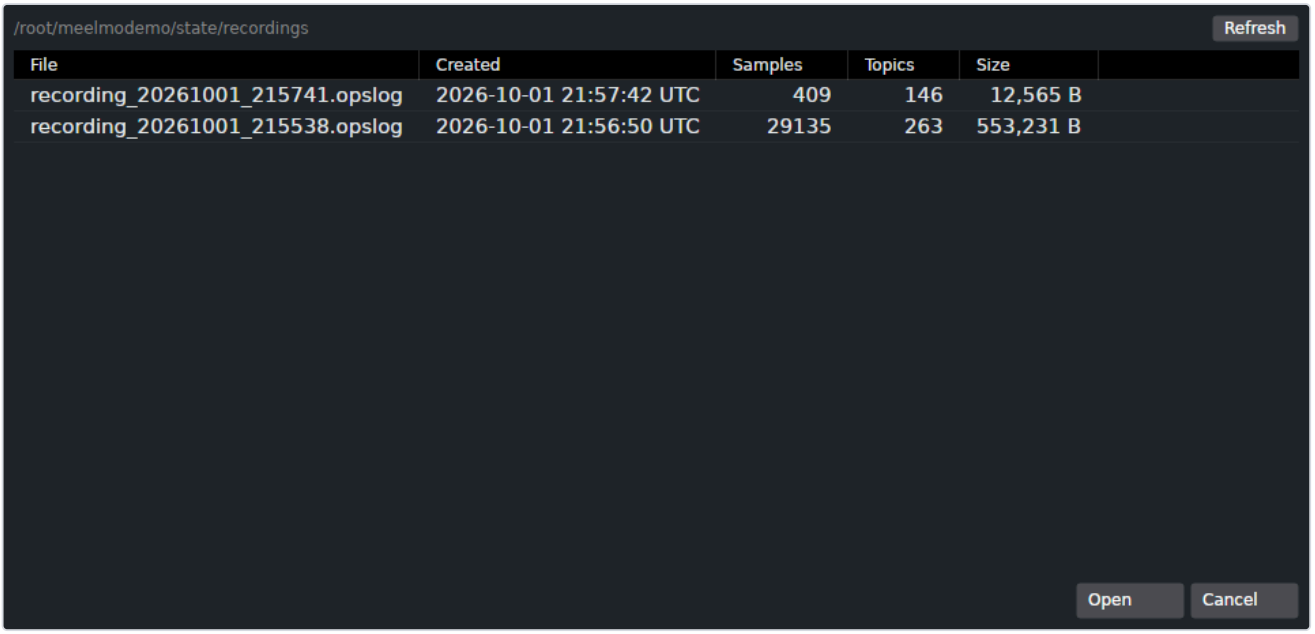
A completed trigger: record while the pressure signal is greater than 8.5, keep recording 30 seconds after the condition clears, and prepend a 10-second pre-trigger buffer. The condition is shown as valid.

Worked example: capture every Loader fault with its run-up

1. Open **Recording ▸ Trigger…** and tick **Enable trigger recording**.
2. In **Signal**, choose `Packaging/Loader/Faulted`. Because it is a boolean, the comparison list offers **is TRUE** and **is FALSE**; choose **is TRUE**.
3. Check that **Compiles to** names `[Packaging/Loader/Faulted]`, and that it says *Condition is valid.*
4. Set **Keep recording after clear (seconds)** to `30`, so the recovery is in the file too.
5. Set **Pre-trigger buffer (seconds)** to `20`, so the 20 seconds before the fault (the Loader's `CollatorBacklog` and `PouchInfeedRate` building up) are included.
6. Click **OK** and start acquisition. Each Loader fault now produces its own file.

Browsing recordings

Recording ▸ Browse Recordings… lists the recordings in the recordings directory, whose path is shown at the top of the window.



The Browse Recordings window listing a recording of the packaging line in the recordings directory: its file name, when it was created, 28,035 samples across 263 topics, and its size

Column	Meaning
File	The file name.
Created	When the recording was started.
Samples	How many samples it holds.
Topics	How many signals it holds.
Size	The file size.

Refresh re-reads the directory (useful while a continuous recording is rolling files). Select a recording and click **Open**, or double-click it, to open it for playback. **Cancel** closes the window.

Playback

Playback puts the Desktop into a mode where the same trend graphs, cursors, statistics and analysis tools work over a recording instead of over the plant.

To open a recording:

- **File ▸ Open Recording…** (or the main toolbar's open-recording button) and choose an `.opslog` file anywhere on disk; or
- **Recording ▸ Browse Recordings…** and open one from the list.

When a recording opens:

1. The status bar shows **Mode: Playback** instead of **Mode: Live**.
2. The **Signal Tree** is rebuilt from the recording's own signals, so it shows what was recorded rather than what the current project acquires.

3. A fast overview renders immediately. Plot signals by dragging them from the tree as usual.
4. Zooming re-reads the file for a finer slice rather than magnifying what is on the screen, so zooming in reveals individual samples.
5. The chart toolbar's **Play** button runs the recording.

In playback:

- **File ▸ Export Plotted Signals to CSV…** reads the plotted signals from the recording itself.
- **File ▸ Export Timing Evidence (CSV)…** is enabled (see below).
- You can open a recording that is still being written, which is what you want during a long capture.

File ▸ Close Recording (enabled only in playback) returns to live.

Playback is a Desktop feature. With the Web HMI module, the browser has a separate history replay that re-animates a published operator page from the historian; it is not the same thing as opening a recording.

Worked example: measuring a recorded fault

1. **Recording ▸ Browse Recordings…**, select the file the trigger in the previous example wrote, and click **Open**.
2. Drag `Packaging/Loader/Faulted`, `Packaging/Loader/CollatorBacklog` and `Packaging/Checkweigher/Starved` onto a trend graph.
3. Zoom around the fault. Because the file had a 20-second pre-trigger buffer, the backlog building up before the fault is visible.
4. Turn on measurement mode and drop **A** on the fault and **B** where the Checkweigher starved. **X2-X1** is how long the buffer held.
5. Add a note (**Annotations ▸ Add note here…**) and save the project.
6. **File ▸ Close Recording** to return to live.

Timing evidence and the recording timing analysis

A diagnosis often depends on whether two signals can really be compared at the millisecond level. Two tools answer that from a recording.

Export Timing Evidence (CSV)

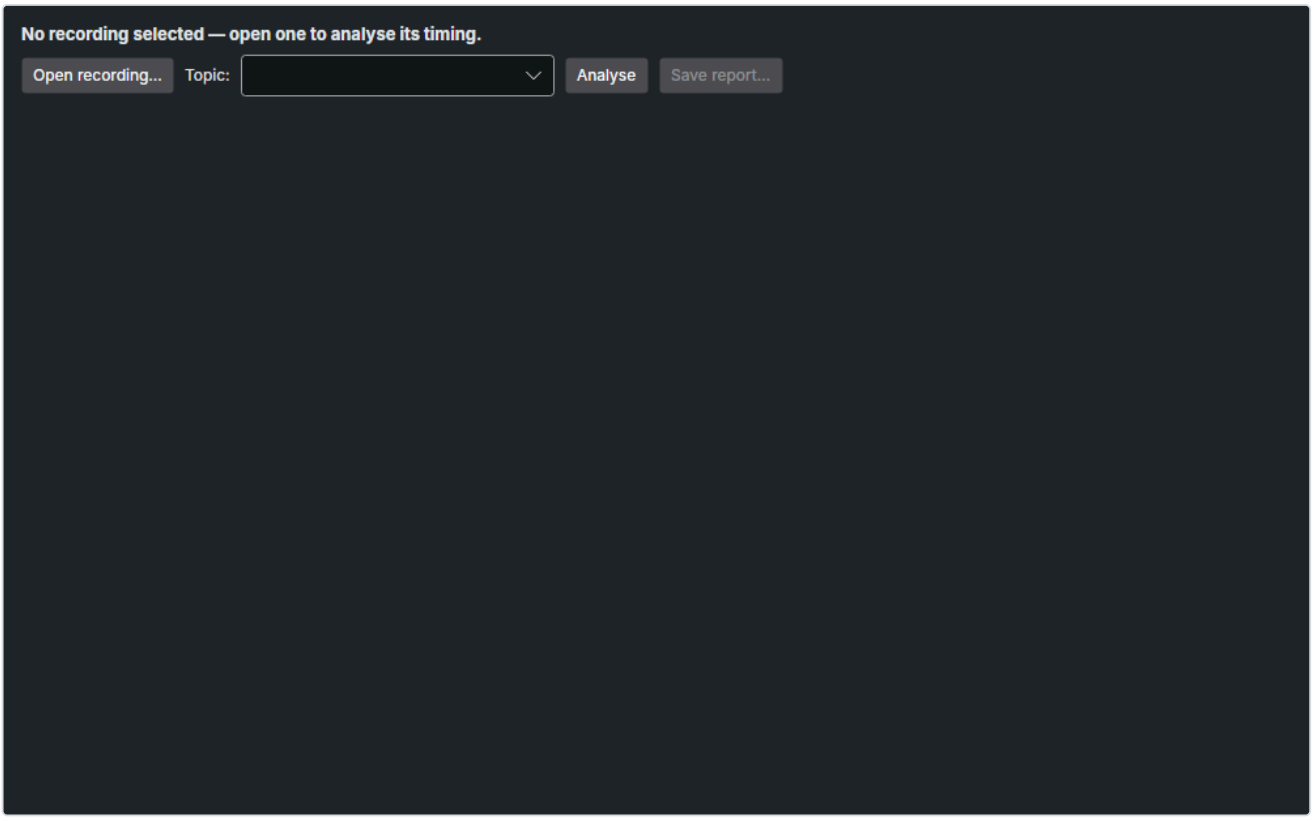
File ▸ Export Timing Evidence (CSV)… is enabled only in playback (otherwise it says *Open a recording first — timing evidence is exported from a recorded file, not from live values.*). It writes one row per sample, for the plotted signals if a chart has any and for the whole recording otherwise, to a file named `<recording>-timing.csv` by default. The columns are:

`SourceTimeUtc, ReceiveTimeUtc, LowerBoundUtc, UpperBoundUtc, HalfWidthMs, Origin, Coherence, Topic, Quality, Value, Sequence`

Samples recorded without uncertainty estimation export with the bound columns empty, not with invented values.

Recording Timing Analysis

Recording ▸ Recording Timing Analysis… reads a whole recording and reports what its timestamps are worth.



The Recording timing analysis window before a recording is chosen: an Open recording button, a Topic list, Analyse, and Save report, which stays disabled until an analysis has run.

1. Click **Open recording…** and choose an `.opslog` file (if a recording is open in playback, it is offered already).
2. In **Topic**, leave **auto (busiest signals)**, or choose one signal to analyse.
3. Click **Analyse**. The window says *Analysing… this reads the whole recording.*
4. Read the report. **Save report…** saves it as a text file for a commissioning record.

The report has these sections:

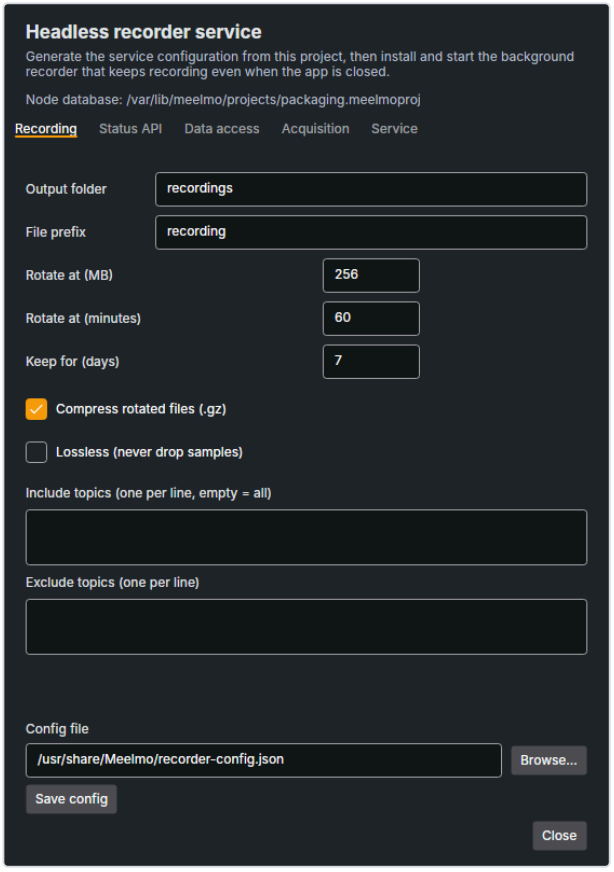
Section	What it tells you
Inferred controller scan	Per signal: samples, changes, change ratio, the inferred controller scan period and its phase bound, and how many scans were skipped.
Interval coverage against the reconstructed scan grid	How often each sample's reported time interval covers the reconstructed scan instant, the mean error, and the resolution of the reconstruction. Coverage is only meaningful to that resolution; poll faster to tighten it.
Cross-group interval consistency	For pairs of signals in different groups: when two reads returned the same value of a fast-moving signal they saw the same scan, so their intervals must intersect. Violations are counted with the worst gap.
What this does and does not establish	A reminder that both techniques observe the controller through the link being tested: they can falsify the timing model but cannot certify it.

Attach the saved report and the timing evidence CSV when somebody asks whether two signals in a diagnosis could in fact be compared. The live equivalent of this analysis is **Administration ▸ Diagnostics…**, described in the *Diagnostics and support* chapter.

The recorder service

The recorder service is a headless recorder that runs as a Windows service, so a machine records (and can serve acquisition and data access) without anybody signed in and without the Desktop open. **Recording ▸ Recorder Service…** generates its configuration from the current project and installs and controls the service.

The window (*Headless recorder service*) shows the node database the service will use and has five tabs, with a **Config file** path, **Browse…** and **Save config** at the bottom. The configuration is a machine's, not a plant's (which port, which certificate on this box), and is kept in `recorder-config.json` in the data folder.



Recording ▸ Recorder Service…, the Headless recorder service window on its Recording tab: the node database it was generated from, then Output folder recordings, File prefix recording, rotation at 256 MB or 60 minutes, seven days kept, compressed rotated files, Lossless cleared and empty Include and Exclude topics; the Config file path with Browse… and Save config at the foot

Recording tab

Field	Meaning
Output folder	Where the service writes recordings (default `recordings`).
File prefix	The start of each file name (default `recording`).
Rotate at (MB), **Rotate at (minutes)**, **Keep for (days)**	Rotation and retention, as in Recording settings.
Compress rotated files (.gz)	Compress rolled files.
Lossless (never drop samples)	Lossless capture.
Include topics	One signal per line; empty records all.

Field	Meaning
Exclude topics	One signal per line, left out.

Status API tab

Status API (read-only remote monitoring). Exposes `GET /health` and `GET /status` over HTTP so that **Recorder Status…** (below) or a monitoring system can watch the service.

Field	Meaning
Enable the status API	Turns it on.
Bind host	`localhost` by default (loopback only).
Port	The HTTP port (default `5077`).
Bearer token	Required when the bind is reachable from the network.
Allow an unauthenticated bind on a routable interface (not recommended)	Overrides the token requirement.

A line under the fields says what the settings mean: *The status API is disabled.*, *Loopback only — reachable from this machine.*, or *Reachable off-box and token-protected.* For encrypted access, keep it on localhost and put a TLS reverse proxy in front.

Data access tab

Data-access API (REST, WebSocket and gRPC). Serves history and live values from the service, the process that acquires. Fields: **Serve the data-access API**, **Listen on**, **Port**, **gRPC port (0 shares the API's)**, **Certificate (PKCS#12)**, **Certificate password**, **State directory**, **Trusted reverse proxies**, **Serve gRPC beside REST and WebSocket**, **Serve the MCP surface…** and **Serve a routable address without TLS (not recommended)**. A routable address with no certificate is refused at start-up, and the tab says so before you save. Callers authenticate as the node's service accounts. This API is used by the data-access and SDK features, which are described separately where an installation includes them.

Acquisition tab

Acquisition server (desktop clients). Lets Desktop workstations attach to the acquisition the service is already running, as views onto it rather than additional pollers.

Field	Meaning
Serve acquisition to desktop clients	Turns it on.
Listen on	The address (default `127.0.0.1`).
Port	The carrier port (9701 by default for a node host).
Certificate (PKCS#12), **Certificate password**	Required to serve the network.
Also accept clients on this machine over the local transport	Local clients without TLS.
Require clients to present a certificate as well	Mutual TLS.

Attaching clients sign in with the node's accounts.

Service tab

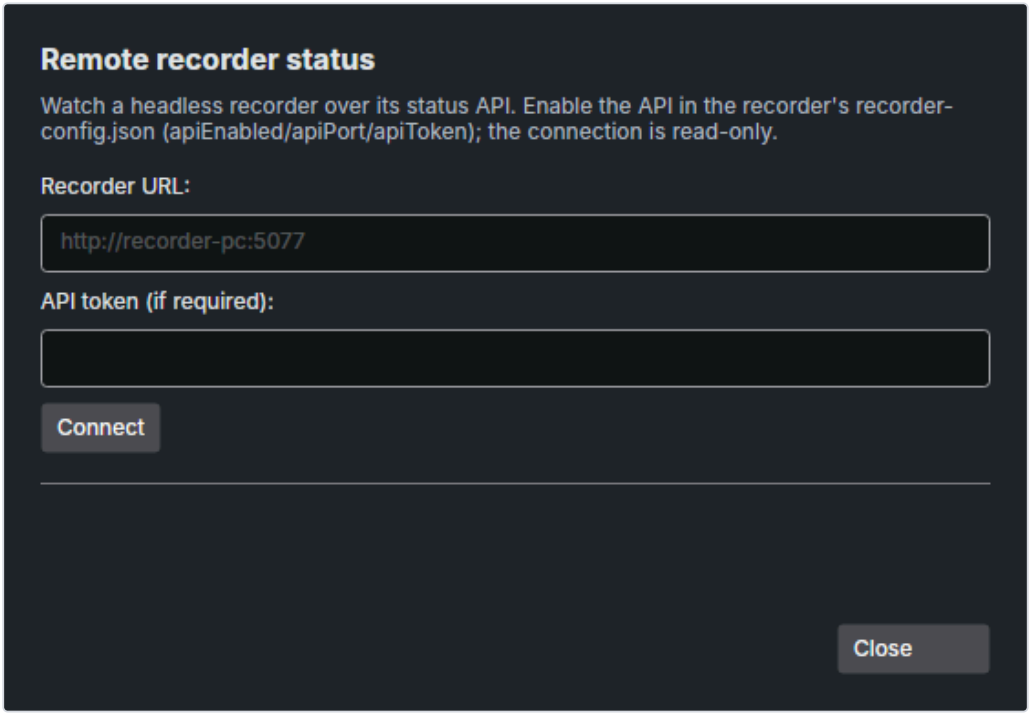
On Windows: the **Recorder program** path, a **Status** line (**running**, **stopped**, **not installed**) with **Refresh**, and **Install**, **Start**, **Stop** and **Uninstall**. On Linux and macOS the tab says to use the systemd unit instead.

Worked example: record the packaging line unattended

1. Open the packaging project and choose **Recording ▸ Recorder Service…**.
2. On **Recording**, set **Output folder** to `D:\Recordings\Packaging`, **File prefix** to `packaging`, **Rotate at (minutes)** to `60`, **Keep for (days)** to `14`, and tick **Compress rotated files (.gz)**.
3. On **Status API**, tick **Enable the status API**, leave **Bind host** at `localhost`, and leave **Port** at its default, `5077`.
4. Set **Config file** to the data folder's `recorder-config.json` and click **Save config**.
5. On **Service**, click **Install**, then **Start**. **Status** reads **running**.

Watching a remote recorder

Recording ▸ Recorder Status… opens *Remote recorder status*: *Watch a headless recorder over its status API. Enable the API in the recorder's recorder-config.json (apiEnabled/apiPort/apiToken); the connection is read-only.*



The Remote recorder status window, asking for the recorder's URL (for example http://recorder-pc:5077) and its API token if one is required.

1. Type the **Recorder URL**, for example `http://recorder-pc:5077`.
2. Type the **API token (if required)**.
3. Click **Connect**.

The window then shows the recorder's state, **Version**, **Uptime**, the **File** it is writing, **Samples** (with the number **dropped**), its **Trigger**, and its **Alarms**. If it cannot connect it says *Not reachable*, *The recorder answered with HTTP …* or *The response was not a recorder status.*

Recording files (.opslog)

Property	Detail
Extension	`.opslog`; rolled files of a continuous recording are compressed (`.gz`) when compression is on.
Contents	Every sample with value, quality, source and receive timestamps, sequence number where available, and (in current recordings) the source-time uncertainty.
Self-describing	The signal tree is rebuilt from the file; no project is needed to open it.
Crash-safe	A session that ends unexpectedly does not cost you the file.
Readable while written	A recording in progress can be opened for playback.
Index	A node with no historian reads its continuous recordings as history, and keeps a `.opscol` index beside each recording for that: its samples again, grouped by signal, with the times a scan stamps on all its signals kept once. About two to four bytes a sample, typically a third to a half the size of the compressed recording; rebuilt if deleted (an index written by an earlier version is rebuilt the same way), deleted with its recording, and never needed to open the recording.
Size	About 16 bytes per sample. At the packaging demonstration's rate of roughly 280 samples a second, that is about 16 MB an hour before compression.

A recording is more important than the application that wrote it: copy `.opslog` files off the machine as you would any other plant record.

Alarms

An alarm tells people that a condition on the plant needs attention. In Meelmo an alarm is defined by a **rule**: a condition over live signals, a severity, and two delays that decide when it raises and when it clears. The node evaluates the rules at acquisition rate; raised alarms appear in the **Alarms Panel**, can be acknowledged and shelved, can be sent by email and other channels, and are recorded in the node's history.

This chapter covers writing alarm rules (**Acquisition ▸ Alarms…**, every field), severities, notifications, the alarm list, acknowledging, shelving, alarm history, and a worked example using the seven rules of the packaging demonstration line.

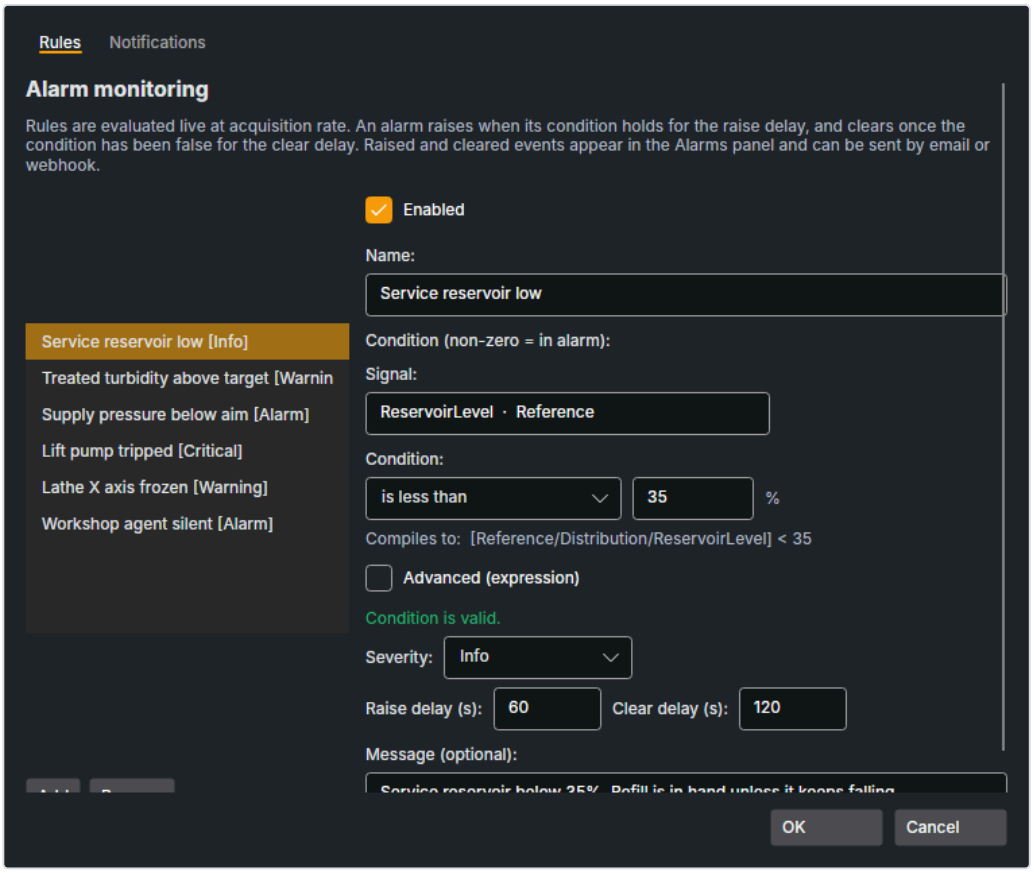
How alarms work

- **Rules belong to the node.** The rules are part of the node's configuration. Saving them activates a new configuration revision on the node, so they are drafted, compared, rolled back and audited like any other configuration (see **Acquisition ▸ Node history…** in the *Diagnostics and support* chapter).
- **The node evaluates them, once.** Rules are evaluated live, at acquisition rate, by the node, not by each workstation. The same engine runs in the Desktop and in the headless recorder service, so a rule does not behave differently because nobody is watching. One condition crossing one threshold is one occurrence with one identity, and an acknowledgement anywhere is an acknowledgement everywhere.
- **Delays decide raise and clear.** An alarm raises only once its condition has held for the **raise delay**, and clears only once the condition has been false for the **clear delay**.
- **A client workstation is a view.** A Desktop attached to a plant server shows the node's alarms; it does not run an alarm engine of its own.

Alarm rules: Acquisition ▸ Alarms…

Acquisition ▸ Alarms… (or the bell button on the main toolbar) opens the alarm window. Configuring alarms needs the permission to configure alarms, which an Engineer or Administrator holds; the window is refused to other roles with *Your role (…) is not permitted to: ConfigureAlarms*.

The window has two tabs, **Rules** and **Notifications**. Its heading, *Alarm monitoring*, repeats the rule: *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.*



The Rules tab of the Alarms window on the packaging line: the seven rules listed on the left with their severities, and the Pouch loader faulted rule open on the right with its signal, comparison, severity Critical, a 1-second raise delay, a 5-second clear delay and its message.

The rule list

The left-hand list shows every rule as *name [severity]*, for example `Pouch loader faulted [Critical]`. A disabled rule is prefixed with `(off)`. Select a rule to edit it on the right.

Button	What it does
Add	Adds a rule called *New alarm* with severity **Warning**, a raise delay of `0` and a clear delay of `5`.
Remove	Deletes the selected rule.
OK	Saves every change to the node.
Cancel	Discards every change made in the window.

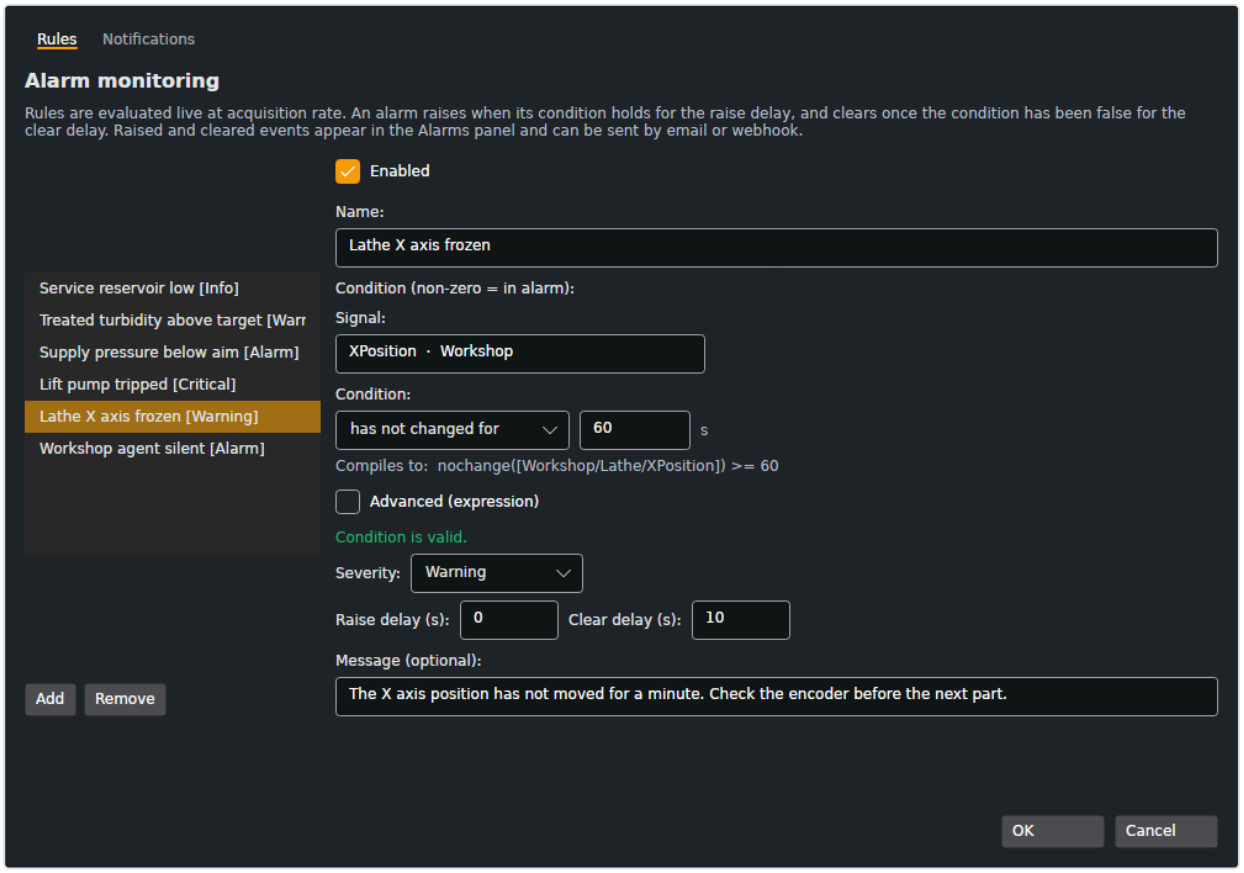
Rule fields

Field	Meaning
Enabled	Whether the rule is evaluated. Untick it to keep a rule without it raising.
Name	The short name shown in the alarm list, for example `Label stock low`.
Condition (non-zero = in alarm)	What puts the alarm on. Built with the three fields below, or typed as an expression.
Signal	The signal the condition tests. Pick from the list of acquired and virtual signals; its unit is shown beside the value.

Field	Meaning
Condition	The comparison. For a number: **is greater than**, **is greater than or equal to**, **is less than**, **is less than or equal to**, **equals**, **does not equal**, **is inside range**, **is outside range** (the range comparisons show **Value** and **Value 2**, joined by *and*). For a boolean: **is TRUE**, **is FALSE**.
Value	The threshold.
Compiles to	The expression the guided fields produce, for example `[Packaging/Labeler/LabelStock] < 15`. Where several machines have a tag of the same name (every station on the packaging line has a `Faulted`), the Signal list shows the full id beside the name, so pick by the id. A stored condition the guided fields cannot write back exactly, such as `[Packaging/Loader/Faulted] > 0` on a Boolean tag, opens in **Advanced (expression)** as it was written, so opening a rule and pressing **OK** never changes it.
Advanced (expression)	Type the condition as an expression instead. Any expression works; non-zero means in alarm. Use `[connection/group/tag]` for a signal and the same functions as virtual signals, for example `abs([Virtual/Computed/WeightDeviation]) > 15`.
(validation)	*Condition is valid.*, or *Invalid:* with the reason. A rule whose condition cannot be compiled is disabled when the node loads it, and the event log says *Alarm '...' disabled - bad condition: ...*.
Severity	**Info**, **Warning**, **Alarm** or **Critical** (see below).
Raise delay (s)	How long the condition must hold before the alarm raises. `0` raises on the first sample.
Clear delay (s)	How long the condition must be false before the alarm clears.
Message (optional)	The sentence an operator reads. If empty, the name is shown instead. Write what is happening and what it will cause, not the threshold.

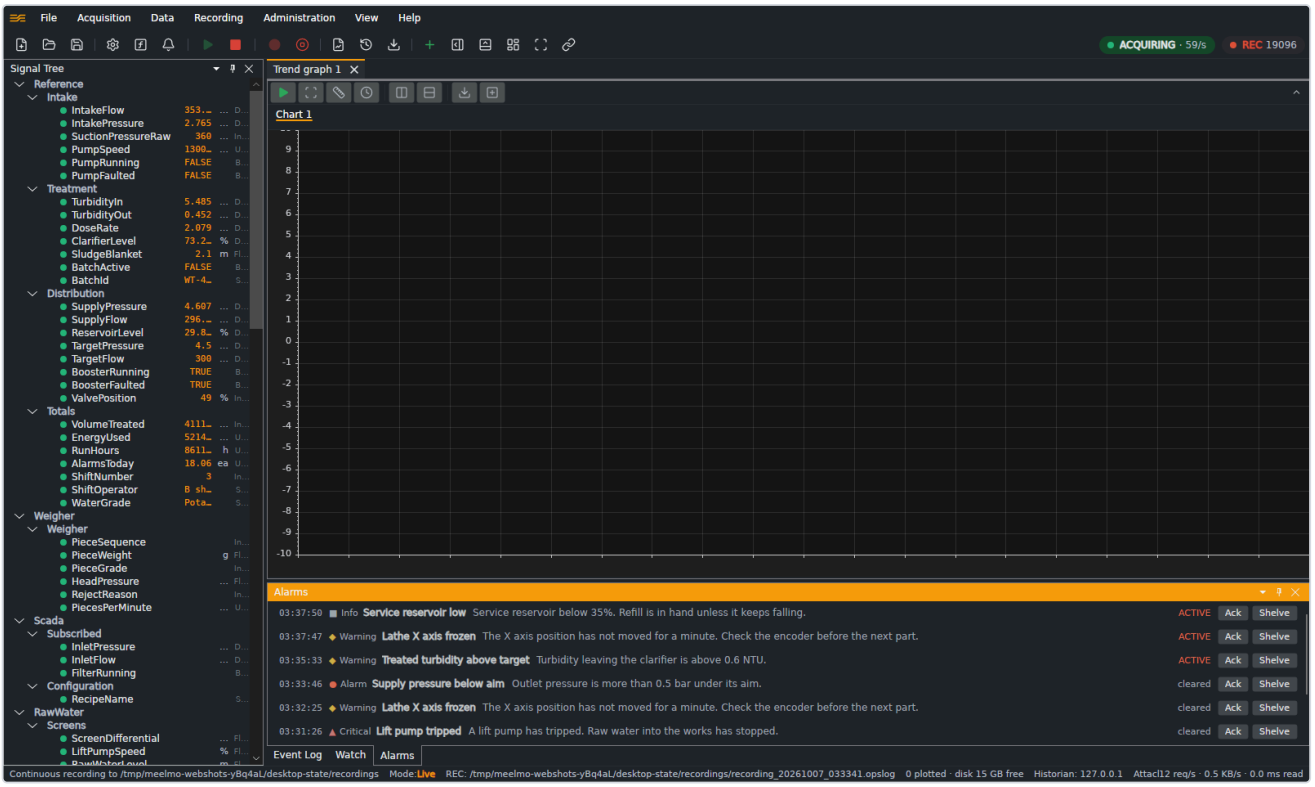
Alarms on silence and on a frozen value

Two comparisons are about time rather than value. **has had no new sample for** raises when the signal has produced no new sample for the stated seconds — the source stopped, or keeps re-sending a value under the same timestamp. **has not changed for** raises when its value has not changed for that long while the reads go on — a register or a sensor frozen at its last value. They compile to `nosample([signal]) >= seconds` and `nochange([signal]) >= seconds`, measured on the node's own clock, so a source that has stopped entirely still raises; nothing raises in the first seconds after a start.

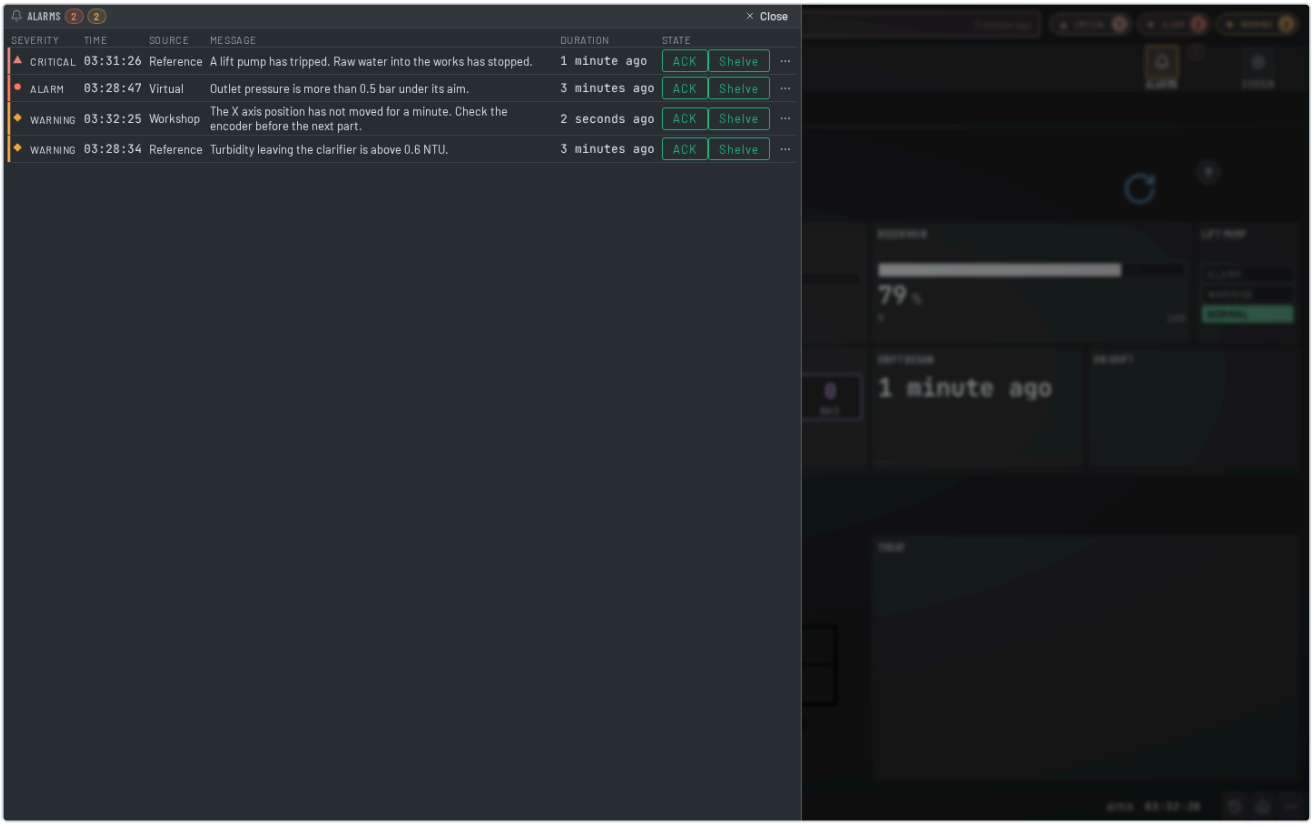


The alarm rule Lathe X axis frozen on the reference works: signal XPosition of the Workshop lathe, Condition has not changed for 60 s, compiling to nochange([Workshop/Lathe/XPosition]) >= 60, severity Warning; Workshop agent silent, a rule on no new sample from the lathe's agent, is beside it in the list

When the lathe's X axis encoder sticks, the agent goes on reporting its last position every half second; a minute later the node raises **Lathe X axis frozen**, and it stands on every client of the node at once — the Desktop's **Alarms** panel and the browser's alarm drawer alike, whichever screen each has open.



The Alarms panel of a Desktop joined to the reference works, docked under the trend: Lathe X axis frozen at the top, a Warning reading The X axis position has not moved for a minute. Check the encoder before the next part, ACTIVE; below it Service reservoir low and Treated turbidity above target active, and an earlier Lathe X axis frozen, Supply pressure below aim and Lift pump tripped returned to normal and marked cleared, each row still offering Ack and Shelve



The browser's alarm drawer open over the works' overview: a Warning from source Workshop, The X axis position has not moved for a minute. Check the encoder before the next part, raised six seconds ago, listed with the works' other standing alarms — the lift pump tripped (Critical), outlet pressure under its aim (Alarm, from a virtual signal) and turbidity above target — each with ACK and Shelve

Creating a rule from a signal

Right-click a tag in the **Signal Tree** and choose **New alarm from this signal…**. The Alarms window opens with a new rule already pointed at that signal (with the boolean comparisons offered if it is a boolean). The same permission applies.

Use the delays

The raise and clear delays are what stop an alarm list becoming noise:

- A **raise delay** stops a signal that crosses a threshold for one scan from waking somebody. `Line not producing` on the packaging line waits 20 seconds, because the Loader stopping for a few seconds is normal.
- A **clear delay** stops a value sitting on the threshold from flapping between raised and cleared. Every demonstration rule clears only after 5 seconds of being false.

A rule with neither delay is the usual cause of an alarm list that nobody reads.

Severities

Severity	Mark in the alarm list	Use it for
Critical	▲	Something that stops production or endangers equipment, needing action now.
Alarm	●	An abnormal condition that needs an operator's action soon.
Warning	◆	Something that will become a problem if left, such as a consumable running low.
Info	■	A condition worth knowing about that needs no action.

Severity is shown in the alarm list by a glyph and the severity's name as well as by colour, so it can be read by somebody who cannot tell red from amber. The same glyphs are used by the browser (with the Web HMI module). Severities are ordered, so the **Minimum severity** of a notification sends that severity and everything above it.

Notifications

The **Notifications** tab sends raised alarms to people outside the application.

Field	Meaning
Minimum severity	Only alarms at this severity or above are sent.
Also notify when cleared	Send a second message when the alarm clears.
Send email (SMTP)	Send by email, with **SMTP server**, **Port**, **Use TLS**, **User**, **Password**, **From** and **To (comma-separated)**.
Post JSON webhook	POST each alarm as JSON to **Webhook URL**.
Post to Slack	Post to an **Incoming webhook URL**.
Post to Microsoft Teams	Post to a **Workflow URL**. Use a Teams Workflows (Power Automate) URL; the older Office 365 connector was retired and accepts messages without showing them.

Field	Meaning
Page through PagerDuty	Using an **Integration (routing) key**; **Resolve the incident when the alarm clears**.
Page through Opsgenie	Using an **API key** and **Region** (`us` or `eu`); **Close the alert when the alarm clears**.

Raise and clear are correlated by a key, so a flapping condition updates one incident rather than opening a new one each time.

The SMTP server is the node's mail relay. The server, port, TLS, sign-in and sender set here are used by the node for every mail it sends: scheduled reports that ask to be mailed go through it too, whether or not **Send email** is ticked for alarms. Clearing the server therefore stops report mail as well. The password is sealed by the node when it is saved and opened only by the node when it sends; the field shows *stored on the node — type to replace*, or *not set*, and **Clear** removes it.

Worked example: email the shift supervisor

1. On **Notifications**, set **Minimum severity** to **Alarm**.
2. Tick **Also notify when cleared**.
3. Tick **Send email (SMTP)** and fill in **SMTP server** `smtp.plant.local` , **Port** `587` , tick **Use TLS**, and give **User** and **Password**.
4. Set **From** to `meelmo@plant.local` and **To** to `shift.supervisor@plant.local` .
5. Click **OK**.

With the packaging rules, `Pouch loader faulted` (Critical), `Line not producing` , `Case weight out of tolerance` and `Compressed air low` (Alarm) are now emailed; the three Warning rules are not.

The alarm list

Raised and cleared alarms appear in the **Alarms Panel** (**View ▸ Alarms Panel**), newest first. It keeps the most recent 200 rows. Each row shows:

Column	Meaning
Time	When the alarm raised (local time, `HH:mm:ss`).
Severity	The glyph and the severity's name.
Name	The rule's name.
Message	The rule's message.
State	**ACTIVE** (raised and not acknowledged), **ACKED** (raised and acknowledged), or **cleared**.
Ack	Shown while the alarm is active and not yet acknowledged.
Shelve	Opens the shelving window with this alarm's rule selected.

While alarms are active, the status bar shows the count and the highest severity, for example `2 ALARMS [Critical]` .

If the panel is not being told about the node's alarms, it says so in a band at its top rather than showing an empty list that would look like a healthy plant. For example: *This workstation is not watching the node's alarms: it has no data credential for that node, so nothing can ask what its rules are holding. Open Deployment and save, which asks the node for one. Live values are unaffected.*, or *The node's alarm state is unavailable: … The panel is showing what it last knew.*

On a client workstation the panel shows the node's alarms, polled from it. An alarm that raises and clears between two polls would be seen by neither, so the panel also sweeps what the node recorded and reports anything that opened and closed unseen. That sweep needs the workstation's credential to carry the `ReadEvents` permission; if it is missing the panel says *The node's alarm history is not being swept: … Alarms that raise and clear between updates may not be announced.* Re-saving the deployment (**Administration ▸ Deployment…**) mints a credential that has it.

With the Web HMI module, operators see the same alarms in the browser's alarm drawer, and an acknowledgement in either place is seen in both. A web host that reads other plants (linked nodes) shows their standing alarms in its drawer too, under each plant's prefix; it never acknowledges or shelves them — that is done on the plant the alarm belongs to.

Acknowledging

Acknowledging says "I have seen this". It does not clear the alarm: an alarm clears only when its condition has been false for the clear delay.

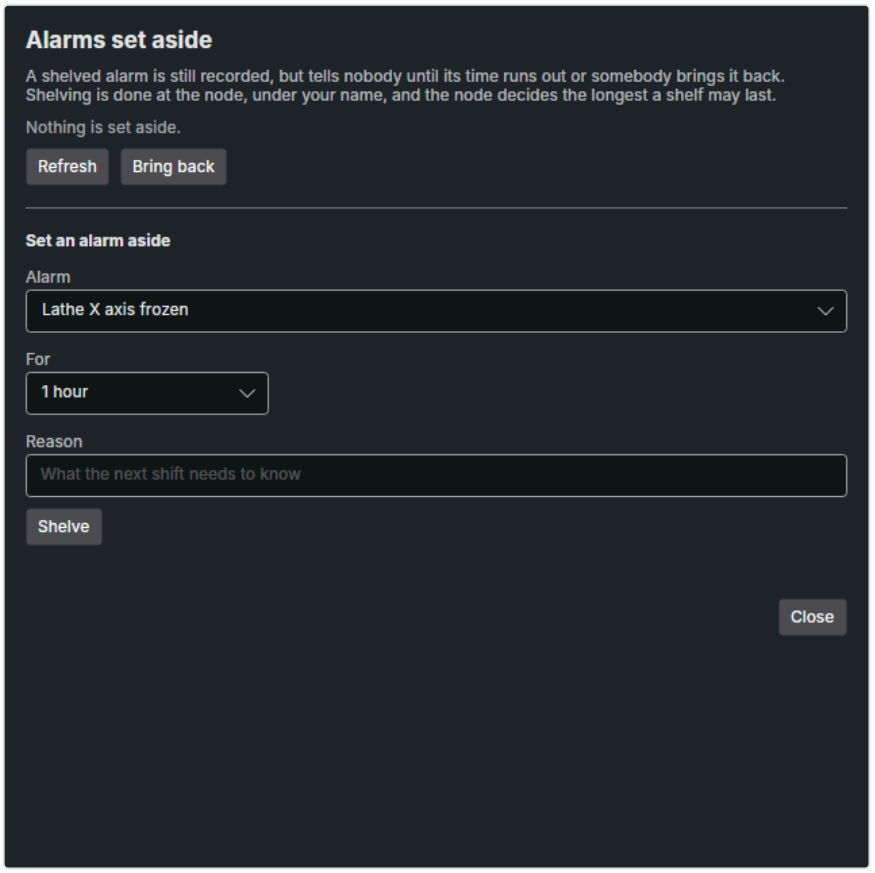
1. Find the row in the **Alarms Panel**. Its state is **ACTIVE**.
2. Click **Ack**.
3. The node checks that you may acknowledge (the permission to acknowledge alarms, which an Operator holds), records the acknowledgement against the occurrence under your name, and answers. The row's state changes to **ACKED** when the node has confirmed it.

The acknowledgement is made **at the node, as you**, both on the workstation that holds the node and on a client workstation attached to a plant server. If the node refuses, the status bar and the event log say why, for example *The node did not acknowledge '…': You are not permitted to do this.*, *The node no longer holds this alarm.*, *Your session at the node has ended. Sign in again.* or *The node did not answer.*

Shelving

Shelving sets an alarm aside for a while, with a reason, when it is known to be irrelevant: a sensor under repair, a planned film change. A shelved alarm is still recorded; it tells nobody until its time runs out or somebody brings it back.

Open the shelving window with **Acquisition ▸ Shelved alarms…**, or with **Shelve** beside an alarm in the Alarms Panel (which selects that alarm's rule for you).



The Shelved alarms window: the list of alarms set aside (empty here) with Refresh and Bring back, and below it the form to set an alarm aside with the alarm, the duration and a reason. The message at the bottom shows a refusal because the session at the node had ended.

The window, titled *Alarms set aside*, has two parts.

The list of shelved alarms shows every alarm set aside at the node, each as *Set aside by … · returns at … (… left)*, or *Nothing is set aside.*

Button	What it does
Refresh	Asks the node again.
Bring back	Ends the selected shelf early; the alarm reports normally again ('…' *is back.*).

Set an alarm aside shelves a rule:

Field	Meaning
Alarm	The rule to shelve. Shelving applies to the rule, so it covers the next occurrence and the ones after it.
For	`15 minutes`, `30 minutes`, `1 hour` (the default), `2 hours`, `4 hours` or `8 hours`.
Reason	Required: *What the next shift needs to know*.
Shelve	Asks the node to shelve it. The answer is '…' *is set aside until …*, or *Not done:* with the reason.

The node decides the longest a shelf may last (eight hours unless the site set another maximum) and shows the time it actually granted, which is shorter than you asked for when you asked for more. Shelving needs the same permission as acknowledging, is written in the node's audit trail under your name, and lasts across a restart of the process that holds the node.

Worked example: shelving during a film change

The wrapper's film reel is being replaced, and `Stretch film low` will stay raised until the new reel is loaded.

1. In the Alarms Panel, click **Shelve** on the `Stretch film low` row. The shelving window opens with **Alarm** set to `Stretch film low`.
2. Set **For** to `30 minutes`.
3. In **Reason**, type `Film reel change on the wrapper, WO-84122`.
4. Click **Shelve**. The list shows *Set aside by shift · returns at 14:30 (30 minutes left)*.
5. When the reel is loaded early, select the row and click **Bring back**.

Alarm history

Every alarm occurrence is recorded by the node in its event store as an event of type `Alarm`: when it raised, when it cleared (the same record is completed, not a second one added), when and by whom it was acknowledged — kept when it clears afterwards — and, for an alarm raised while its rule was shelved, until when and by whom. Who acknowledged or shelved it is also written in the node's audit trail. The record's identity is derived from the rule and the moment it raised, so a clear finds its raise even across a restart. This is the record that the data-access API, the browser (with the Web HMI module) and a client workstation's sweep all read, so every screen describes one alarm the same way.

The Desktop's Alarms Panel shows the recent occurrences; the headless recorder's status (**Recording ▸ Recorder Status…**) also reports its alarms. For an alarm that needs investigating, plot the rule's signals over the time of the occurrence on a trend graph, or read them from the historian with **Data ▸ Browse History…**.

Alarm performance: Acquisition ▸ Alarm performance…

How the alarm system performed over a period, measured the way ANSI/ISA-18.2 (its monitoring and assessment clause) and EEMUA Publication 191 measure it, from the alarm history above. Choose **Last 24 hours**, **Last 7 days** or **Last 30 days**, and an **Area** (all areas, or one), and press **Analyse**. The same review is on the browser's *Analytics* ▸ *Alarms* page, on the API (`GET /alarms/performance`), for an assistant (`alarms_performance`) and as a report source, and all of them say the same thing about the same period.

The **findings** come first, **worst first**: each is one plain sentence and one suggested action, with its verdict — **Unacceptable**, **Manageable**, **Acceptable** or **For information**. The **key figures** follow with the guidance each was held to, then the **top contributors**.

Figure	Benchmark	Verdict
Average alarms per hour given to the operator	ISA-18.2, EEMUA 191: about 6 very likely acceptable, about 12 the most manageable	≤ 6 acceptable, ≤ 12 manageable, more unacceptable
Average per 10 minutes	about 1 acceptable, about 2 the most manageable	≤ 1, ≤ 2, more
Most alarms in 10 minutes	EEMUA 191: under 10 manageable, more than 100 definitely excessive	≤ 10, ≤ 100, more
10-minute periods in flood (more than 10 alarms)	ISA-18.2: about 1 % or less	≤ 1 %, ≤ 5 % (this product's band), more

Figure	Benchmark	Verdict
Share of the 10 most frequent alarms	ISA-18.2: about 1 % to 5 % at most	≤ 5 %, ≤ 20 % (this product's band), more; not judged with ten alarms or fewer
Chattering alarms (3 or more raises within a minute)	target none	any is unacceptable
Fleeting alarms (cleared within 2 s, twice or more)	target none	any is unacceptable
Stale alarms (standing over 24 h) on any day	ISA-18.2: fewer than 5	none acceptable, 1–4 manageable, 5 or more unacceptable
Priority mix (high = Critical, medium = Alarm, low = Warning and Info)	about 80 % low, 15 % medium, 5 % high	high ≤ 7.5 % and medium ≤ 25 % acceptable; ≤ 15 % and ≤ 40 % manageable
Shelved, out of service, time to acknowledge, time to return to normal	no number in the guidance	reported (median and 90th percentile), not judged

What it names, and what to do:

- **Top contributors** — the ten alarms carrying most of the load, with their share. An alarm with 5 % or more is named as a *bad actor*: decide whether it needs an operator each time.
- **Chattering** — with a suggested **clear delay** long enough to bridge nine in ten of the quiet moments between a clear and the next raise (rounded up to 1, 2, 3, 5, 10, 15, 30, 60, 120 or 300 s), or a deadband on its limit. The rule's present clear delay is shown where the window knows it.
- **Fleeting** — with a suggested **raise delay** just longer than its longest fleeting raise.
- **Stale** — standing over a day; fix, re-set, suppress by state, or take out of service under a management of change.
- **Floods** — runs of 10-minute periods over 10 alarms, with the alarm that started each.
- **Duplicate and consequential pairs** — two alarms that raised within 5 seconds of each other on at least four in five of one's raises, at least five times: duplicates when it holds both ways, otherwise the second follows the first and is a candidate for suppression while the first stands.
- **Areas** — each area's own rate, where more than one area raised.

Shelved alarms (raised while their rule was shelved) are counted apart and not in the rates, because nobody was shown them. **Limits:** the node records no operator position, so the per-position figures the standards describe are given **per area** — the connection an alarm's condition reads. *Out of service* is what the record can show: an occurrence ended because a revision removed or disabled its rule, and a rule disabled now; how long a rule was disabled is not recorded. An alarm's priority is its severity. A review reads at most 100,000 occurrences and says when the period held more.

Worked example: the packaging line's alarm rules

The demonstration project `examples/demo/packaging.meelmoproj` has seven rules. They show the typical kinds of alarm on a line: a station fault, the line stopping, consumables running low, quality out of tolerance, and a utility.

Name	Condition	Severity	Raise delay (s)	Clear delay (s)	Message
Pouch loader faulted	`[Packaging/Loader/Faulted] > 0`	Critical	1	5	The station that sets the line rate has a fault standing; everything downstream will starve.
Line not producing	`[Packaging/Loader/Running] < 1 && [Packaging/Line/ChangeoverActive] < 1`	Alarm	20	5	The pacemaker is not running and no changeover is booked.
Label stock low	`[Packaging/Labeler/LabelStock] < 15`	Warning	5	5	The label reel is nearly out; the labeler will stop and block the line behind it.
Case weight out of tolerance	`abs([Virtual/Computed/WeightDeviation]) > 15`	Alarm	10	5	Cases are leaving the checkweigher off specification and are being pushed to the chute.
Reject rate above target	`[Virtual/Computed/Quality] < 98`	Warning	60	5	More than two cases in a hundred are being rejected.
Compressed air low	`[Packaging/Line/AirPressure] < 6.0`	Alarm	15	5	Header pressure is below what the sealers and the pusher need.
Stretch film low	`[Packaging/Wrapper/FilmRemaining] < 12`	Warning	5	5	The wrapper will not finish many more pallets on the film it has.

The project's notifications are set to a **Minimum severity** of **Warning** with **Also notify when cleared** on, and no channel enabled.

Points worth noticing:

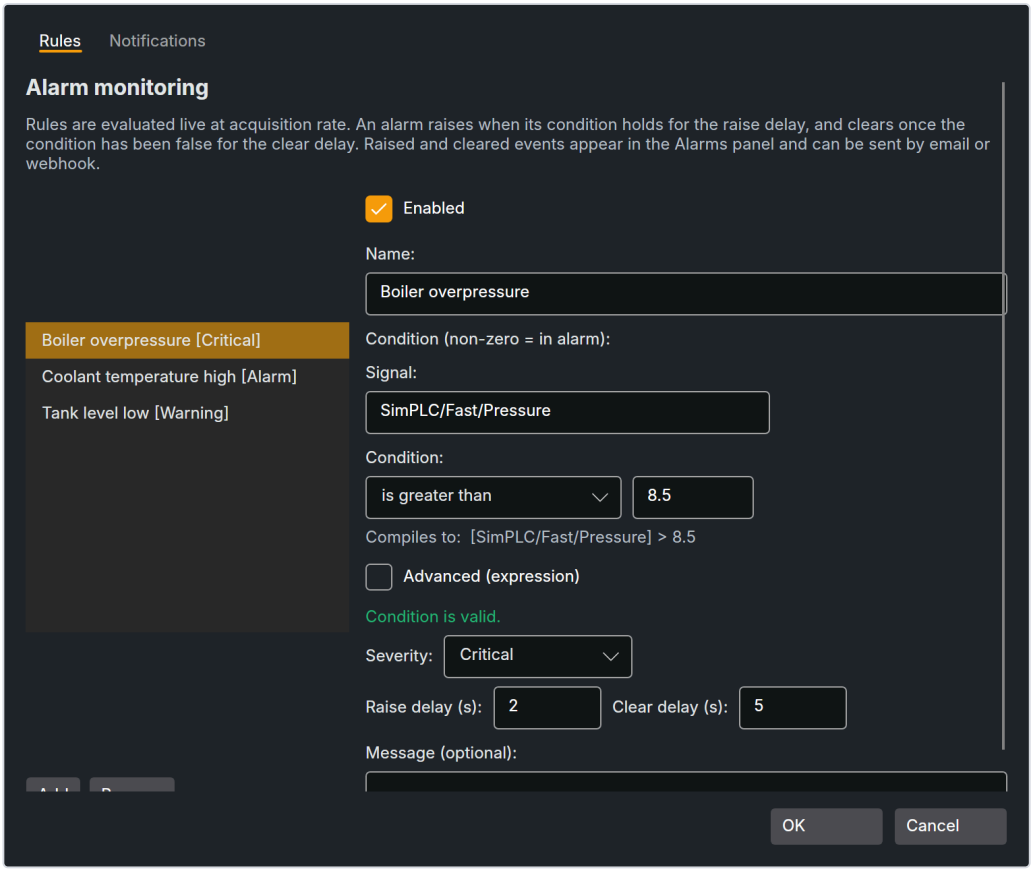
- **Why the delays differ.** A Loader fault is raised after 1 second because it stops the line at once. `Line not producing` waits 20 seconds so a short stop does not raise it, and it excludes changeovers (`ChangeoverActive`) because a booked changeover is not a fault. `Reject rate above target` waits 60 seconds because the reject rate is a ratio that moves with every case.
- **Rules on virtual signals.** `Case weight out of tolerance` tests the virtual signal `WeightDeviation` (`CaseWeight - WeightTarget` , where the target is 1240 g), so the tolerance of ±15 g follows the target if the product changes. `Reject rate above target` tests the virtual signal `Quality` , the percentage of good cases.
- **Consumables as warnings.** `LabelStock` and `FilmRemaining` fall steadily as the reels are used; a Warning at 15 % and 12 % gives the operator time to prepare a change before the machine stops.

- **Messages that say what will happen.** Each message tells the operator the consequence (everything downstream will starve; the labeler will block the line), not the threshold that was crossed.

Building the Label stock low rule step by step

1. Open **Acquisition ▸ Alarms…** and click **Add**.
2. Leave **Enabled** ticked and type `Label stock low` in **Name**.
3. In **Signal**, choose the Labeler's `LabelStock` (unit `%`).
4. Set **Condition** to **is less than** and the value to `15`.
5. Check that **Compiles to** reads `[Packaging/Labeler/LabelStock] < 15` and the window says *Condition is valid.*
6. Set **Severity** to **Warning**, **Raise delay (s)** to `5` and **Clear delay (s)** to `5`.
7. In **Message**, type `The label reel is nearly out; the labeler will stop and block the line behind it.`
8. Click **OK**. The node activates a new configuration revision, and the status line reads *Alarm monitoring active: 7 rules*.

In the demonstration the label stock ramps from 100 % down to 8 % over 15 minutes, so the alarm raises in the Alarms Panel as **ACTIVE** with the ◆ Warning mark roughly every quarter of an hour, and clears 5 seconds after the reel is refilled.



An alarm rule being authored: the Boiler overpressure rule of an earlier example project, raised when the pressure signal is greater than 8.5 for 2 seconds and cleared 5 seconds after it falls back, with severity Critical.

The historian

The historian is where a Meelmo node keeps the plant's history for months and years: every sample it acquires, stored with its timestamp and quality, summarised into rollups so that a year can be drawn as quickly as an hour, and trimmed by a retention policy so that the disk does not fill. This chapter explains what the historian is, how to configure it in **Data ▸ Historian**, what it does when its store is unavailable, how to size the disk for it, how to read history back with **Data ▸ Browse History**, and how to move history from one store to another with the `Meelmo.Migrate` program.

The historian and recordings are different things

Meelmo has two ways of keeping what it acquires, and they are independent of each other. Both can run at the same time, and turning one on or off never affects the other.

	Recording	**Historian**
What it produces	An `.opslog` file per recording, or rotating files in continuous mode	A store: a segmented local database, a SQLite file, or a server database
Started by	**Acquisition ▸ Recording ▸ Start Recording**, a trigger, or continuous recording	Nothing: once turned on, the node historizes everything it acquires
Meant for	A test, a commissioning run, an incident you want as a file to hand to somebody	Months of history for trends, datasets, reports, analysis and the data-access API
Summaries	None; playback reads the file	Rollup tiers (100 ms, 1 s, 1 min) built as it goes
Retention	Rotation size, age and retention of files (**Recording settings…**)	**Raw retention (days)** on the historian's **Ingestion** tab
Read with	**File ▸ Open Recording…** (playback)	**Data ▸ Browse History…**, datasets, reports, analysis, the API

The historian is **off** until somebody turns it on. That is deliberate: an installation that is upgraded must not start writing gigabytes to a disk nobody sized for it. The window says it in its first line: "*Where this installation keeps history. Recording to .opslog files is separate and keeps working whatever is set here.*"

A node with no historian serves its recordings as history

A node that keeps no historian is not a node with no history, if it records continuously. The process holding it (the recorder service on a plant server, or the Desktop while it records continuously) tells the node where its recordings are, and the node serves its history from them: a browser trend opened on a page shows the window behind *now*, as far back as the recordings are kept, rather than starting empty. The recorder says so once when it starts:

> "*This node has no historian; history is read from its recordings in /var/lib/meelmo/recordings (as far back as they are kept).*"

What this gives and what it does not:

- **Everything that reads the node's history** reads it as it reads a store: browser trends, the data-access API and what is built on it (datasets, reports, a client workstation's history), and automation replays. It answers with raw readings, or with buckets (minimum, maximum, first, last, average) at

the width the point budget buys. The Desktop's own **Data ▸ Browse History…** reads a historian only. Where nothing was recorded (the recorder was stopped, a file is missing, a trigger was not capturing) the answer has a gap, never a line drawn across it.

- **It is read-only.** The recordings are not changed. Beside each one the node keeps an index, a file with the same name ending in `.opscol`, which holds the recording's samples again grouped by signal so that ten signals out of twelve thousand can be read without reading the other eleven thousand nine hundred and ninety. It is built in the background as the recorder writes, typically takes a third to a half of the space of the compressed recording (two to four bytes a sample: the times one scan stamps on all its signals are kept once, and values are kept as the decimals or floats they are), is rebuilt if deleted or written by an earlier version, and is deleted with its recording. The first time a node reads a directory that already holds days of recordings it indexes them newest first, so recent history is available within seconds and older history over the following minutes.
- **There are no rollups.** A long window is summarised from the recorded samples each time; for months of history, or for history that must outlive the recordings' retention, turn on a historian.
- **A historian takes over.** Turn one on and history is served from it from the next request; the recordings are no longer read. Turn it off again and the recordings are served once more. Nothing is copied between them.

> **Not the same as Node history. Acquisition ▸ Node history…** lists the node's *configuration revisions* (what was activated, when and by whom) and lets an engineer roll back to one. It has nothing to do with sample history; it is covered in the *Configuration and deployment* chapter.

How signals are identified

Every signal is stored under a **stable hierarchical id**, the same path you see in the signal tree, for example `Packaging/Checkweigher/CaseWeight` (connection, then acquisition group, then tag). The id never depends on a database key. Renaming a signal creates an alias, so its history stays reachable under both the old and the new name.

When the node was not acquiring

The historian writes down when each connection's acquisition started and stopped, beside its signals, as `<connection>/$Acquisition` (1 where a session started, 0 where it stopped), and `<connection>/$Heartbeat` every ten seconds while it runs. Every read then reports the time a signal was not being acquired as a **gap**: the node or its host stopped, acquisition stopped, a connection disabled or restarted by a revision. Trends, history replay, the Desktop and report charts break their lines there instead of drawing a straight line across it, and dataset alignment carries no value across it.

- A node that ended without a stop (killed, crashed, power lost) is found by its next start: the gap runs from the later of its last heartbeat and the signal's last reading to the restart. A value sent only when it changes is therefore held at most until the last heartbeat, not until the restart.
- A store written before these markers existed reports what it always did, and so does the time before a store's first marker: nothing was written down about it, so nothing is claimed. The first restart after an upgrade is therefore not reported as a gap.
- The node keeps a session of its own, `$Acquisition` and `$Heartbeat` at the root of what it writes, from the moment it opens its store until it closes it. It covers the signals that belong to no connection — the node's `Health/…` measurements, computed and model signals — so a node that was down is a gap in those too. Computed signals also follow acquisition as a whole.
- The markers are the historian's bookkeeping, not plant signals: no catalogue, signal picker, search, dataset, export, OPC UA address space, broker namespace or publication lists them, for any account. The history read uses them by id.

- A store's record of its sessions is cached for a minute per process; a stop or start this node writes clears the cache at once. A second node writing the same database is seen within the minute.

Where the historian runs

The historian belongs to the **node**, the process that acquires. On a standalone workstation the node runs inside the Desktop; on a server it runs in the recorder service. In both cases there is exactly one store for the plant.

A Desktop attached to a node elsewhere (a *client* workstation) keeps no history of its own. It reads the node's history over the node's data API, and when you open **Data ▸ Historian** it shows and changes the **node's** configuration. When you save from a client, the message "*applied at the node*" confirms that the change was made on the node, which opens the store itself.

The status bar shows one line about the historian whenever one is configured:

Status line	Meaning
(nothing)	No historian is configured. Not a fault. A node that records continuously serves its recordings as history (see *A node with no historian serves its recordings as history*).
`Historian: embedded` (or the provider name)	Running: samples reach the store.
`Historian: … — spooling N sample(s), spool P% full`	**Degraded**: the store is not taking data and the spool is absorbing it. Nothing is lost yet.
`Historian: … — the spool cannot be written to`	Degraded, and the spool's own disk refuses writes. Loss starts here.
`Historian: unavailable — <reason>`	The store could not be opened. The reason and the remedy are shown.
`…; N refused sample(s) kept in <path>`	The store accepted the connection but refused a batch; it was kept in a dead-letter file (see below).
`… (N source(s) unavailable)`	A store read alongside the historian cannot be reached.

Screens that read history (**Browse History**, datasets, reports, analysis) are available while the historian is Running or Degraded. When it is off or unavailable they refuse with the reason, for example "*No historian is configured for this installation. Turn one on in the historian settings to keep history beyond the current session.*"

Providers: where history is stored

The **Provider** box on the **Store** tab chooses the storage technology. The base product includes two local stores; three server databases are sold as separate modules.

Provider	Needs	Use it for
Embedded historian	Nothing	The default. A local segmented store: no database, no server, no configuration.
SQLite	Nothing	The same data in a single portable SQLite file, readable with standard SQL.
SQL Server	The SQL Server historian module and a connection string	Reporting against your own SQL Server estate

Provider	Needs	Use it for
Oracle	The Oracle historian module and a connection string	The same, on Oracle
TimescaleDB	The TimescaleDB historian module and a connection string	Time series at scale
QuestDB	The QuestDB historian module, a host and its two ports	Fast time series on a site that runs QuestDB. Holds samples and rollups only: the context store stays `journal`
ClickHouse	The ClickHouse historian module, a host and its HTTPS port	A site that runs ClickHouse for analytics. Holds samples and rollups only: the context store stays `journal`

A provider that is sold but not installed or not licensed on this machine is still listed, marked *not available here*, with the reason beneath it, for example: "*Not installed on this machine. The database stores are an optional component: run the installer again and tick "Database history providers"*." The SQL Server, Oracle, TimescaleDB, QuestDB and ClickHouse stores are described in their own chapters, present on installations licensed for those modules.

The embedded historian

The embedded historian is a columnar store written for time series. It needs no native dependency and no configuration:

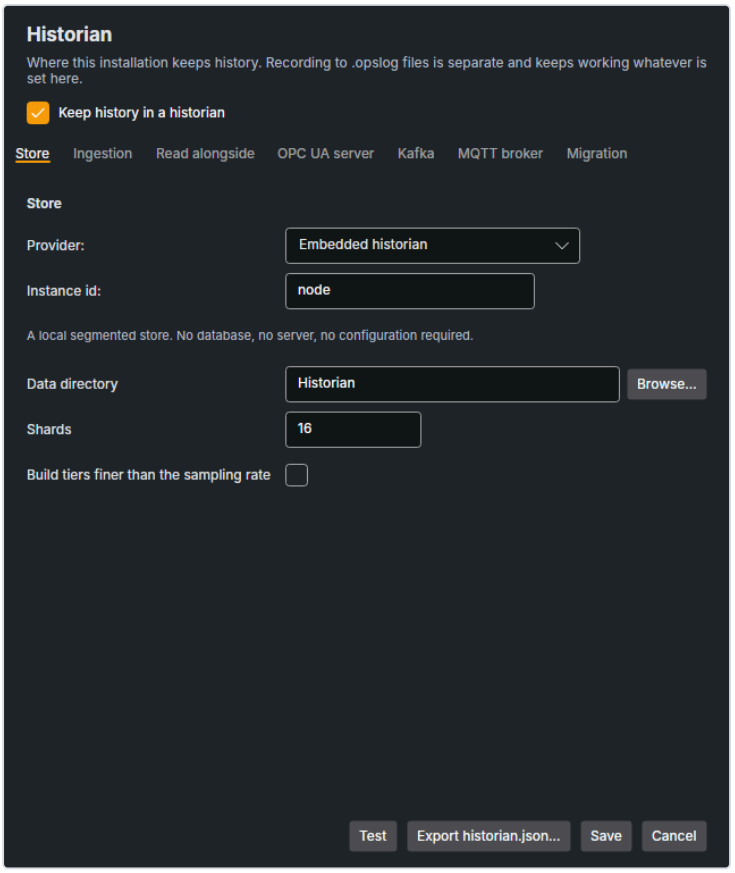
- Data is split across **shards** (16 by default) by a hash of the signal id, so several signals are appended in parallel.
- Every write is appended to a **write-ahead log** and flushed to disk before it counts as stored. After a crash the log is replayed and de-duplicated, a torn tail is truncated and a corrupt block skipped, so recovery is automatic.
- Buffered samples become a **block** (about 2,000 samples or one minute), blocks become a **segment** (about 128 MB or six hours), and a sealed segment is never modified again.
- Blocks are compressed per column (timestamps, values, quality). A sealed segment holds roughly **12.9 bytes per sample**.
- Only one node may write to a store. A second process pointed at the same folder is refused at start rather than allowed to corrupt it.

SQLite

SQLite keeps everything in one file (`historian.db` by default). It is slower than the embedded store for large volumes but can be copied as one file and opened with any SQLite tool, and it offers the read-only reporting views described under *Migration tab* below.

Data ▸ Historian: the configuration window

Data ▸ Historian… opens the **Historian Settings** window. Opening it needs the permission to configure the historian (the Engineer and Administrator roles); an operator who tries is told so. The window has a heading, the **Keep history in a historian** check box, seven tabs, and a row of buttons that apply to all tabs.



The Historian window on the packaging demonstration node: the embedded historian is selected on the Store tab, with its data directory, shard count and sparse-tier option

Element	What it does
Keep history in a historian	The master switch. Off: nothing is historized and every field below is disabled. On: the node writes everything it acquires into the store chosen on the **Store** tab.
Store tab	Which provider and its own settings. See *Store tab* below.
Ingestion tab	The write queue, the spool, retention and rollups. See *Ingestion tab* below.
Read alongside tab	Detection rules file, other stores queried with the historian, and publication to an MQTT or Sparkplug B broker. See *Read alongside tab* below; publication is in the *Industrial integration* module's chapter.
OPC UA server tab	Serves this node's signals read-only to OPC UA clients. See the *Industrial integration* module's chapter.
Kafka tab	Produces samples, alarms and events to a Kafka cluster. See the *Industrial integration* module's chapter.
MQTT broker tab	Runs a governed MQTT broker on this node. See the *Industrial integration* module's chapter.
Migration tab	Writing to a second store for a cutover, and reporting views. See *Migration tab* below and *Moving history between stores*.
Test	Asks the node whether it can reach the configured store with these settings, and whether the store's schema is ready. The answer appears at the bottom of the window.
Create schema	Appears only after **Test** found the store reachable but without the historian's tables. See *Creating a schema* below.
Export historian.json...	Saves this configuration as a file, for example to set up an identical node. Passwords in it are sealed by this node, so on another installation you must enter them again.

Element	What it does
Save	Validates every tab and applies the configuration to the node, which reopens the store.
Cancel	Closes without changing the running configuration.

Nothing you change takes effect until you press **Save**; closing the window with **Cancel** leaves the node exactly as it was. **Test** and **Create schema** are always answered by the **node**, not by the workstation the window is open on, because the node is the machine that holds the credentials and will open the store every day afterwards. A Desktop that is not attached to any node is told "*This installation is not attached to a node, so there is nothing that can reach the database.*"

Store tab

Field	Meaning
Provider:	The storage technology (see *Providers* above). Changing it discards settings the new provider does not use, and the window lists what was discarded.
Instance id:	Names this historian instance. It appears in provenance (where a value came from) and in migration checkpoints, so two instances holding different data must never share a name. A node's own store is `node`.
(description line)	The provider's own one-line description, for example "*A local segmented store. No database, no server, no configuration required.*"
(provider settings)	Generated from the provider. See the tables below.

Embedded historian settings

Field	Default	Meaning
Data directory	`Historian`	Where segments, the catalog and the write-ahead logs are stored. A relative path is relative to the node's data folder. **Browse…** picks a folder.
Shards	`16`	Parallel append streams, rounded up to a power of two. Leave at 16 unless support advises otherwise.
Build tiers finer than the sampling rate	Off	Off: a rollup tier that would hold only one sample per bucket for a signal (for example the 100 ms and 1 s tiers of a signal sampled once a second) is not built for it, and a read builds those buckets from raw samples, with the same result. On: every tier is kept for every signal, which costs disk but makes wide trends of slowly sampled signals cheaper. Tiers are always built while raw samples have a retention period, because the tiers may outlive the raw data.

SQLite settings

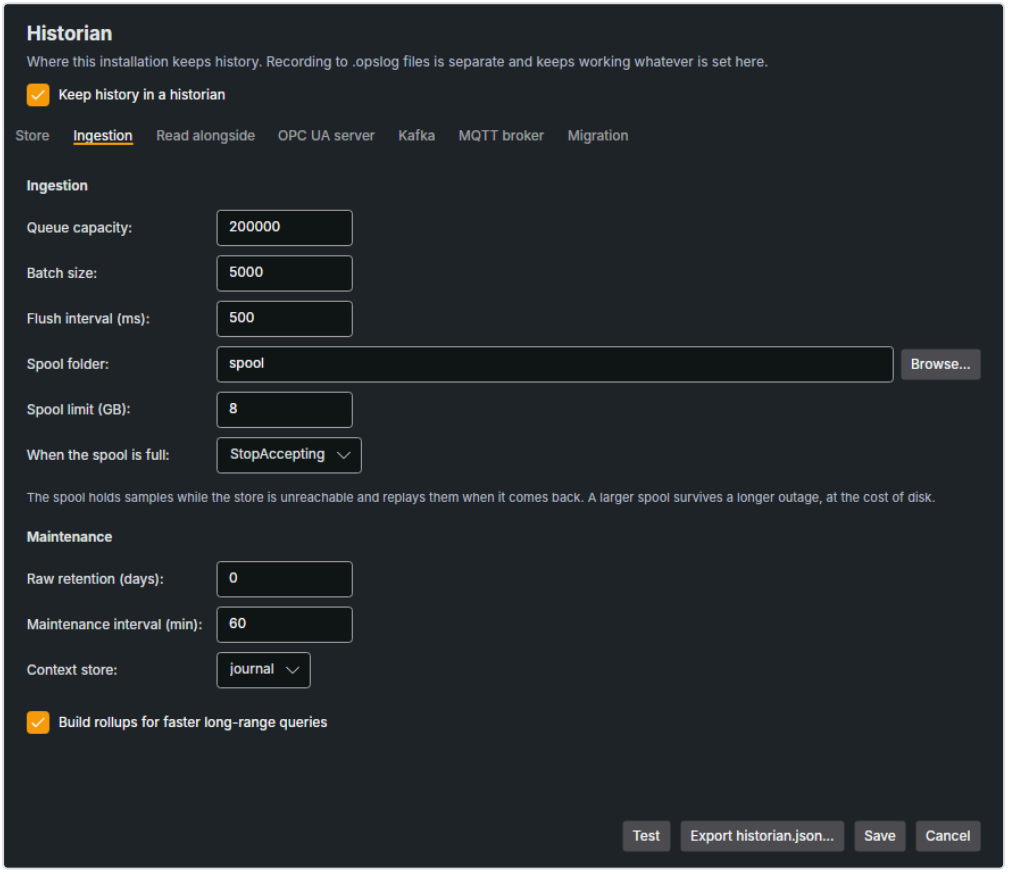
Field	Default	Meaning
Database file	`historian.db`	The SQLite file. Created if it does not exist.
Insert batch size	`10000`	Rows per insert transaction.
Command timeout (s)	`120`	How long a single database command may run.

A server provider shows its connection string, password, schema and batch settings here instead. The password is sealed and stored by reference, never written into a settings file; a password box showing *stored — leave blank to keep it* already has one.

Copy existing history into the new store appears below the provider settings when this Desktop runs the node and the store it writes to now holds history. It is ticked. If the save moves history to a different store — another provider, or another file or database — the node copies what the old store held into the new one in the background after the switch; a save that leaves the store where it is copies nothing. See *Copying history when you switch stores* below.

Ingestion tab

The **Ingestion** tab has two sections: how samples travel to the store, and the maintenance that keeps the store fast and bounded.



The Ingestion tab of Data ▸ Historian with its defaults: queue capacity 200000, batch size 5000, flush interval 500 ms, spool folder spool, spool limit 8 GB, StopAccepting when the spool is full; under Maintenance raw retention 0 days, maintenance every 60 minutes, the journal context store and Build rollups ticked

Ingestion

Field	Default	Meaning
Queue capacity:	`200000`	Samples the in-memory queue between acquisition and the store may hold. Acquisition hands a batch to this queue in microseconds and never waits for the store.
Batch size:	`5000`	Samples per write to the store.
Flush interval (ms):	`500`	How long a partial batch waits before it is written anyway. This is roughly the delay between acquiring a sample and being able to read it from history.
Spool folder:	`spool`	Where samples are held on disk while the store is unreachable. Relative to the node's data folder. Empty disables spooling, which means an outage costs data instead of disk; that is reasonable only for the embedded store, which cannot be unreachable.

Field	Default	Meaning
Spool limit (GB):	`8`	The ceiling on the spool. Keep it at 8 GB, and never below 64 MB.
When the spool is full:	`StopAccepting`	`StopAccepting` refuses new samples for this store (and counts them) while everything else keeps running. `OverwriteOldest` discards the oldest spooled samples to make room. The default is the one that loses nothing already held.

Below the fields the window reminds you: "*The spool holds samples while the store is unreachable and replays them when it comes back. A larger spool survives a longer outage, at the cost of disk.*"

Maintenance

Field	Default	Meaning
Raw retention (days):	`0`	Days of raw samples to keep. `0` keeps everything.
Maintenance interval (min):	`60`	How often retention and rollup building run. Neither happens unless something schedules it; this is that schedule.
Context store:	`journal`	Where the semantic model (runs, events, states the detection rules find) is kept. `journal` : a checksummed journal beside the history, answered from memory; right for almost everyone, up to hundreds of thousands to low millions of entities. `provider` : in the historian's own database, for very large sites or for one that wants the model in the same backup as the samples. The embedded store has no such database, so choosing `provider` with it is refused as a configuration error.
Build rollups for faster long-range queries	On	Builds the rollup tiers. Turn it off only if the provider builds its own or the history is small enough that reading raw samples is always acceptable.

Compression tab

The **Compression** tab decides whether this store may keep **compressed** history, and how much each signal may be simplified. It is off by default; see *Compressing stored history* below for what it does and what it promises.

Field	Default	Meaning
Compress stored history (lossy, within each signal's stated bound)	Off	The switch. Until it is ticked nothing is compressed, whatever the rules or the project's tags say.
Rules	none	For the signals the project's tags, groups and connections do not describe. Each rule names its **Signals** by a pattern over the signal id (`*` for any run of characters, `?` for one: `Line1/*` , `*/Temperature` , `*`); the first rule that matches a signal applies.
Compression deviation / **or % of span**	—	How far the stored line may stray from what reached the door, in the signal's engineering units, or as a percent of a span.
Exception deviation / **or % of span**	none	A change no larger than this from the last value that passed never reaches the door. Keep it below the compression deviation, about half of it. It adds to the bound.
Span low / **Span high**	none	The span a percentage is taken of. A tag's write limits stand in when the project gives it none.

Field	Default	Meaning
Shortest interval (s)	none	No value is stored sooner than this after the last one. It gives up the bound for the values it skips, so leave it empty unless a source chatters.
Longest interval (s)	none	A heartbeat: a value is stored at least this often, even when the signal sits flat.

Preview tries the fields above on one signal's own history before anything is saved: type the signal's id, choose the last hour, day or week, and press **Preview**. The answer says how many of the readings would have been kept, the ratio, the largest error measured against every reading and the bound the settings promise — "*Would keep 1,719 of 86,400 readings (50.3:1). Largest error measured: ±0.5; the bound is ±0.5.*" Nothing is written. **Compressed signals** lists every signal being compressed with its ratio, stored and offered counts since its compression last changed, and its bound; **Refresh** reads it again. Both are the node's to answer, so they work where the window is open on the machine running the node; elsewhere the tab says so.

In the browser the same settings, preview and list are a panel of **Administration ▸ Historian**, saved with that page's **Save**.

Read alongside tab

This tab holds three sections.

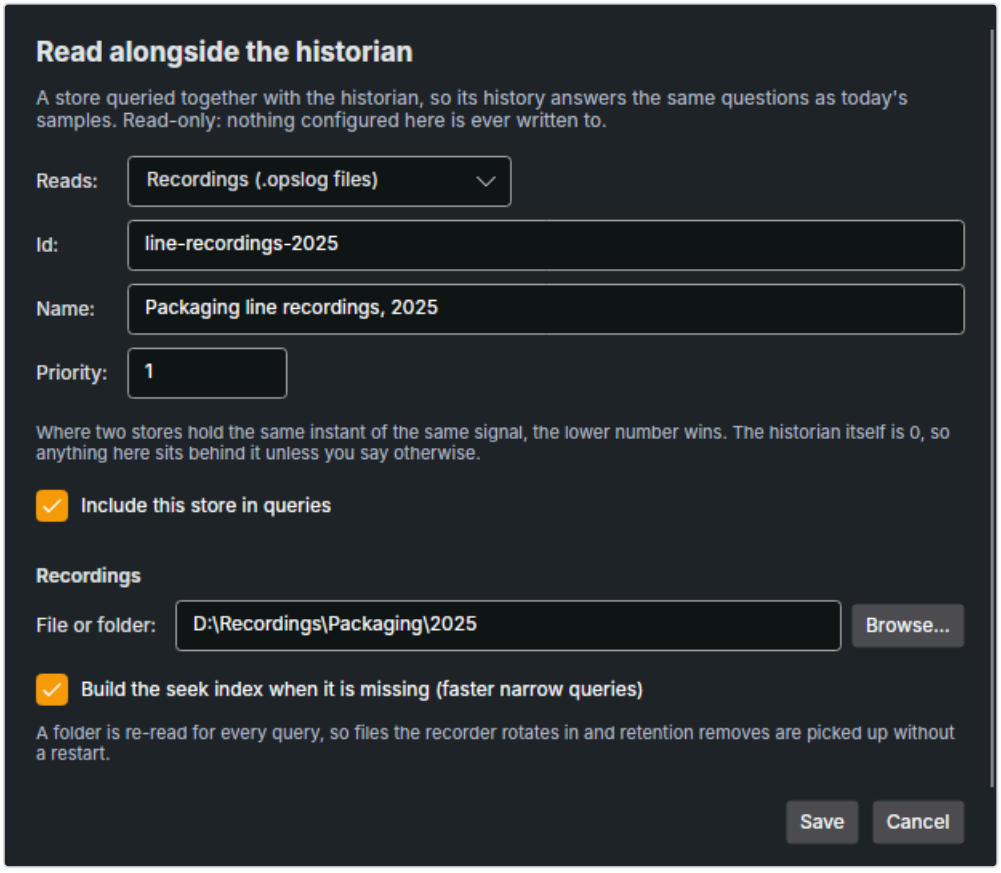
Contextualization. Detection rules file: names the file that holds the node's detection rules (default `semantic-rules.json`, relative to the historian's data folder). Rules turn signal behaviour into runs, events and states; the *semantic model* chapter explains them. With no file, history is recorded with nothing detected from it, which is the right default until the process has been described. You normally edit the rules through **Acquisition ▸ Detection Rules…**, not here.

Read alongside. A list of other stores that are **queried together with the historian**, so old recordings and a database that predates Meelmo answer the same question as today's samples. All are read-only: nothing listed here is ever written. **Add…**, **Edit** and **Remove** manage the list; a store that is switched off shows *(switched off)*. **Add…** opens the **Historical Source** window:

Field	Meaning
Reads:	The kind of store: **Recordings (.opslog files)**, **A Meelmo store kept after a migration**, or **A database this product did not create**.
Id:	Required. It travels with every point this store serves, so a chart can say where a value came from.
Name:	A readable name for the list.
Priority:	Where two stores hold the same instant of the same signal, the lower number wins. The historian itself is `0`, so anything added here sits behind it unless you give it a lower number.
Include this store in queries	Switches the store in or out without deleting its settings.
File or folder: *(recordings)*	An `.opslog` file or a folder of them. A folder is re-read for every query, so files the recorder rotates in, and files retention removes, are picked up without a restart.
Build the seek index when it is missing *(recordings)*	Makes narrow queries into large recordings faster.

Field	Meaning
Provider:, **Connection string:**, **Password:** *(databases)*	Which driver reads the store and how to reach it. The password is sealed and referenced, never written into the settings file.
Schema:, **Signals table:**, **Samples table:**, the id, name, unit, timestamp, numeric value, text value and quality columns, **Value meaning "good":** *(existing database)*	Where the data is in a database Meelmo did not create. Nothing is guessed: required fields are marked with an asterisk; name a numeric column, a text column, or both.

When stores overlap, the merged answer is de-duplicated by signal, source time and sequence, and marked where double-counting is possible.



The Historical Source window opened with Add... on the Read alongside tab, filled in for a folder of the packaging line's recordings: Reads Recordings (.opslog files), Id line-recordings-2025, a name, priority 1, Include this store in queries ticked, the folder and Build the seek index when it is missing

Publication. Publish acquired samples and the broker, transport and topic fields send acquired samples onward over MQTT or Sparkplug B. Publishing is separate from storing. It belongs to the Industrial integration module and is described in that module's chapter; on an installation without it, leave **Publish acquired samples** cleared.

OPC UA server, Kafka and MQTT broker tabs

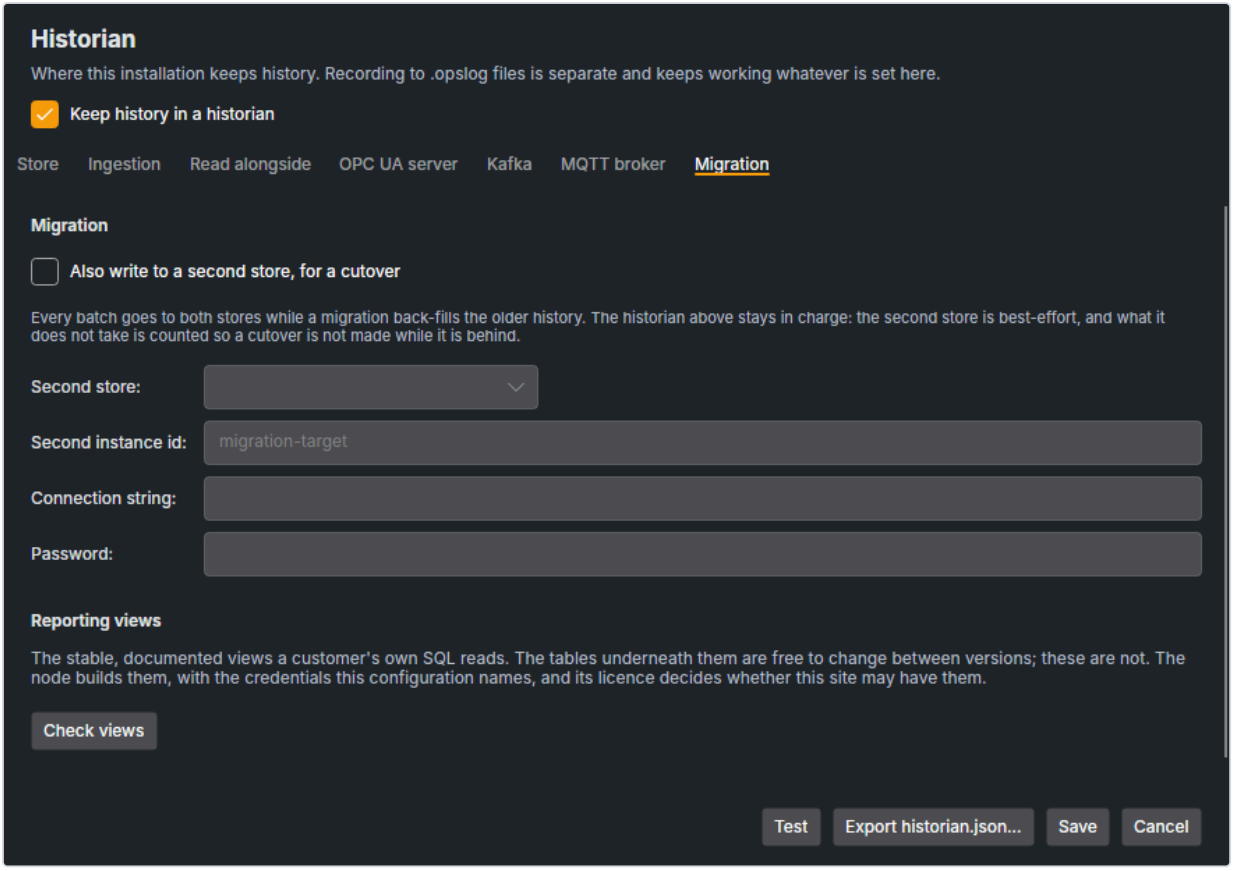
These three tabs are not about storing history. Each one opens a way for other systems to read what the node acquires:

Tab	What it switches on	Off by default	Described in
OPC UA server	**Serve this node over OPC UA**: a read-only OPC UA server on this node.	Yes	the *Industrial integration* module's chapter
Kafka	**Produce to Kafka**: samples, alarms and events produced to a Kafka cluster or to Azure Event Hubs.	Yes	the *Industrial integration* module's chapter
MQTT broker	**Run the broker**: a governed MQTT broker on this node, with access rules and payload contracts.	Yes	the *Industrial integration* module's chapter

They belong to the Industrial integration module. On an installation without it, leave them switched off. Like every other tab, they take effect only when you press **Save**.

Migration tab

Migration. Also write to a second store, for a cutover makes every batch go to two stores while older history is being back-filled into the new one. The store on the **Store** tab stays in charge; the second store is best-effort, and what it does not take is counted, so that nobody cuts over while it is behind.



The Migration tab of Data ▸ Historian: Also write to a second store, for a cutover cleared, so the second store, its instance id, connection string and password are greyed out; below it Reporting views with Check views

Field	Meaning
Second store:	The provider of the store being migrated to.
Second instance id:	Its instance id. Must differ from the first.
Connection string:, **Password:**	How to reach it (database providers only).

The second store receives only what the node acquires from now on. The older history is copied into it by `Meelmo.Migrate`; *Moving history between stores* below gives the order in which to use the two.

Copying history from the previous store. Where a copy asked for on the **Store** tab is running, or has run since the node started, this says how far it has got: the two stores, the instant it copies up to, the share done, the samples copied, the rate and the time left, or why it stopped. **Cancel copy** stops it; **Retry** carries a stopped copy on from where it was, and **Dismiss** forgets it.

Reporting views. Stable, documented SQL views that a customer's own SQL reads (`oa_v_signals`, `oa_v_samples`, `oa_v_rollup_1s`, `oa_v_rollup_1m`, `oa_v_events`, `oa_v_executions`, `oa_v_states`). The tables underneath may change between versions; the views do not. **Check views** asks the node which exist; **Create views…** and **Remove views…** appear after a check. After creating them, grant `SELECT` on these views only, never on the tables. The embedded historian has no SQL underneath it, so a check answers "*This store has no SQL underneath it, so there are no reporting views to have.*" Read the embedded store through datasets, reports or the API instead.

Creating a schema

A new SQLite file or database must have the historian's tables before the node can write to it. Press **Test**: when the store answers but its schema is not ready, the message says so and **Create schema** appears. Pressing it asks for confirmation ("*Create the historian's tables in … ? This writes to that store.*") and then the node creates and checks the tables using the credentials in this configuration.

Creating a schema needs rights that the running service deliberately does not hold (`CREATE TABLE` and `ALTER`). If your site keeps those rights away from the historian's login, sign in with an account that has them, or ask a database administrator to press the button. Afterwards the account can be narrowed to read and write.

Retention and rollups

Rollups are pre-computed summaries of the raw samples in fixed buckets. Meelmo keeps three standard tiers: **100 ms**, **1 s** and **1 min**. Each bucket holds the count, minimum and maximum (with their timestamps), first, last, sum and spread, and good/bad counts, so the average and standard deviation of any longer span can be computed exactly by merging buckets. Rollups are why a year-long trend draws in the time an hour does: a query that asks for 5,000 points over a month reads 1-minute buckets rather than millions of samples.

Every reader (the history browser, datasets, reports, the API) asks for a time window and a **point budget**, and one resolution selector chooses the finest tier whose bucket count fits the budget. A width that is not stored (say 3 s) is built by merging the tier beneath it. Booleans are aggregated as numbers (the average is the share of time they were true); text signals have no numeric form and are always served raw.

Retention removes raw samples older than **Raw retention (days)**. Two rules keep it safe:

- The historian **never deletes raw data before its rollups exist**.
- Retention removes **whole sealed segments**, never individual samples. A segment is deleted only when everything in it is older than the retention window, so raw data slightly older than the setting can remain until its segment ages out.

Rollup tiers are not removed by raw retention: a node that keeps 30 days of raw samples still answers a one-year trend from its 1-minute tier. Retention and rollup building run every **Maintenance interval (min)**.

What a bucket's value is: the mean, or the average over time

A bucket's value is the **mean of the samples in it** for most signals, and the **time-weighted average** for a signal that reports only when it changes. The bucket carries both either way, with its count, extremes, first and last.

The difference matters for anything read by subscription. An OPC UA subscription, and every other report-by-exception acquisition, sends a value only when it moves, so the samples crowd into the minutes the plant was busy. A tank level that sat at 10 for 59 minutes and then rose to 100 with ten quick readings in the last minute has a sample mean of about 92 for that hour. It spent the hour at about **11.5** — and that is what the time-weighted average says: each value counted for as long as it held.

How a value is weighed:

- **It holds until the next reading** of the signal (stepped, the default) — what a subscription, a state, a counter and a running flag all mean. **Linear** ramps from each reading to the next instead, and suits only a continuous measurement sampled faster than it changes; integers and booleans are always stepped.
- **The value from before the bucket counts.** The last reading before the window holds from the window's start until the first reading inside it, so an hour in which a valve did not move still has the valve's position as its average.
- **A reported gap weighs nothing** — time the node was not acquiring (see *When the node was not acquiring*) — and a value is not assumed to have survived it: the hold stops at the gap and resumes only with the next reading.
- **Under a quality filter** (*good only*, *exclude bad*) an excluded reading weighs nothing and ends the good value's hold before it, rather than crediting the good value with time the source said something else. With every quality admitted, bad readings are weighed and the bucket's quality says so.
- A **maximum gap**, when a query sets one, is the longest any value is held.
- A bucket that **no sample fell in** but that a value held through is returned as a bucket with a sample count of nought, the held value, and *synthesized* set — so an hourly report of an on-change signal has every hour.

When it is used. By default, for every signal whose catalogue entry says it reports on change (`acquisition.cadence` = `OnChange`, which a subscription writes), and for every window that reaches history stored compressed — which is always weighed **linearly** over the stretches that were compressed, whatever was asked, because the straight line between stored points is what the bound is a bound on (see *Compressing stored history*). Outside those stretches — history from before compression was switched on, or after it was switched off — the query's own rule applies, stepped unless it asked otherwise, so a window that spans a change is weighed each side by its own rule. A periodic signal keeps the sample mean, which for even sampling is the time-weighted average to within a sample; a signal nothing describes keeps it too. Anything can be asked for by name: **Bucket value** in *Browse History* and on a browser trend, `aggregation` on the API (`twa`, `twstddev`). Asking for **Sample mean** always gets the sample mean. Two trends of one signal on one page may ask for different bucket values; each draws its own.

How it is computed, and what it costs. No store computes it. The query service reads the window's raw samples, the last value before it and the gaps the store reports, and weighs the buckets itself — the same way whichever store answered: the embedded store and its rollup tiers, SQLite, SQL Server, Oracle, TimescaleDB, QuestDB, ClickHouse, a recording read as history, several stores read alongside, or another node. The rollup tiers are not read for it: a time integral stored per bucket would be wrong the moment a late sample landed in the bucket before it. So a time-weighted read costs one raw read of the window. For an on-change signal that is small (a row per change); for a fast signal asked for by name over a long window it is not, and it is bounded at **1,000,000 raw samples per signal per read** — past that, or where retention has already removed the raw samples behind a bucket, the buckets keep their sample

means and the read says why (*not time-weighted: …*) rather than weighing part of the window. Measured on a development machine: a million one-hertz samples in ten-minute buckets took about 3 s weighed against 14 ms for the means from the rollups; an on-change signal of two thousand changes over the same eleven days took 4 ms either way.

A window's statistics, the same way. The statistics of a whole window (`/history/statistics` , the gRPC `GetStatistics` , the MCP tool `history_statistics` , and everything in the node that summarises a range through the query service) follow the same rules as a bucket the width of the window: weighed by name (`aggregation` = `twa`), by default for a signal on change and for a window that reaches compressed history, never for an explicit `average` ; the value carried in from before the window, the gaps, the quality filter and a maximum gap count exactly as they do for a bucket; and every store gives the same answer, because the service weighs it from the same raw read. Both figures travel: `average` and `standardDeviation` stay the sample mean and spread, and a weighed window adds `timeWeighting` (`stepped` or `linear`), `timeWeightedAverage` , `timeWeightedStandardDeviation` and the area and time behind them. Quote the time-weighted one when `timeWeighting` is present. The tank level above: a window from 08:00 to 09:00 has `average` 91.8 and `timeWeightedAverage` 11.5; from 07:45, with 20 carried in from the reading at 07:30, its time-weighted average is 13.2. Where the raw samples behind a window are no longer kept, it keeps its sample mean and `timeWeightingNote` says so.

Compressing stored history

A plant that keeps years of history can keep it in a fraction of the space by storing fewer points of each signal, within an error it states. Meelmo does this optionally, per signal, and only for what the historian keeps long term:

- **Never the recordings.** The `.opslog` recordings and their `.opscol` index stay lossless; they are the forensic record.
- **Never what runs live.** Live values, alarms, advisories, analytics and semantic detection see every sample as it was acquired. Only the historian's own sink compresses.
- **Off by default**, and switched on in the historian's settings (*Compression* tab, or `historian.compression.enabled`). Changing anything applies from that moment; nothing already stored is rewritten.

The method. Swinging-door trending (E. H. Bristol, 1990) behind an exception deviation. The last stored value is a hinge; each later value narrows the range of slopes a straight line from the hinge could take and still pass within the compression deviation of every value since. A value is stored only when the next one could not be reached that way, and then it becomes the hinge. Before the door, a value within the exception deviation of the last value that passed is treated as that value, so noise smaller than it never reaches the door.

The promise. *The line drawn straight between two stored values never differs from any value that was acquired by more than the exception deviation plus the compression deviation.* The door checks the slope to each candidate itself, rather than only whether the door has closed, which is what keeps that true; it is tested as a property over thousands of random signals of every shape. Two things are outside it: a **shortest interval**, which skips values whatever they are, and a sample older than the newest one of its signal, which is stored as it came. Where the exception deviation held a stretch flat and the signal then leaves it, the corner is stored with the held value at its own instant — within the exception deviation of the reading there; every other stored value is exactly what was acquired.

What is never simplified. Booleans, integers (a state, an enumeration, a counter — the type does not say which, so none is treated as analogue), texts, a NaN and every reading that is not good quality are kept on every change: the value before the change and the value after it, so a step and a straight line both draw it exactly. A **change of quality**, and a reading going missing, stores the value before and the value after whatever the deviations say. A connection's **stop** stores what its signals were holding before the stop is written, so nothing is drawn across an outage.

Reading compressed history. Compressed history must be read as a straight line between its stored points — that is what the bound is a bound on — and only compressed history: switching compression on does not make last year's history compressed, and switching it off does not make last week's exact. So the node keeps a **timeline** of each signal's compression — when it took effect and what it was — and every read is described by what applied to the window it read:

- the timeline is `history-compression.json` in the node's historian directory, beside the semantic journal: the node's, not a store's, so it survives a restart and a change of store, and history copied from the previous store into the new one is described by it without anything being copied. A signal's compression begins at the first value its door decides; a change or an end takes effect just after the newest value the old door decided; a change made while the node was stopped is recorded at the signal's first value after it starts. History imported from elsewhere is described as acquired, which is all this node knows of it;
- every read carries the compressed stretches of its window, each with its bound (`compressionIntervals`: `from`, `to`, `method`, `deviation`, `exception`, `maximumError`), with `compression`, `compressionDeviation` and `compressionException` the widest of them, and the sentence `compressionNote`, over REST and gRPC and between nodes. A window wholly compressed says "*Compressed (swinging door): values between stored points are interpolated, at most ±0.5 degC from what was acquired.*"; one partly compressed says which part — "*Partly compressed (swinging door) from 2026-10-30 10:00:00Z: values between stored points there are interpolated, at most ±0.5 degC from what was acquired; the rest of the window is stored as acquired.*" A window from before compression says nothing. A window's statistics carry the same fields;
- a bucket and a window's statistics are **time-weighted by default, and linearly over the compressed stretches**, whatever interpolation the query asked for — a ramp stored as its two ends averages to its middle, not to its first value held to the end — and by the query's own rule elsewhere (see *What a bucket's value is*);
- the catalogue entry says how the signal is stored **now** (`history.compression` = `swingingDoor`, with `history.compression.deviation`, `.exception` and `.maximumError`), since when (`history.compression.since`), or, for a signal stored as acquired again, until when it was compressed (`history.compression.until`);
- trends draw a compressed trace straight between points, never as a step, even where it was set to steps; *Browse History* and the browser trends' tooltips show the sentence, so nobody mistakes an interpolated value for a measurement.

The value a door is holding — the newest one, not stored yet — is added to every read that reaches it, so a trend of a signal sitting flat still reaches now: to a raw read as a reading; to a bucketed read merged into the bucket it falls in (its count, extremes, first and last, sums and spread) or as a bucket of its own where the store has none; and to a window's statistics. It is stored when the node stops, when the store is changed (the retiring store stores it before it closes; the arriving one starts afresh with the next value), and when the signal's compression changes.

Where it runs. In the historian's ingestion worker, per signal, as samples come off the queue — before the store, the spool and a dual write's second store. So every provider benefits the same way (embedded, SQLite, SQL Server, TimescaleDB, QuestDB, ClickHouse, Oracle) and none of them knows it is happening. What is spooled during an outage is already compressed and is replayed as it was decided, never compressed again. Copying history between stores (*Copying history when you switch stores*, `Meelmo.Migrate`) reads one store and writes the other directly, so it moves the stored points and compresses nothing. The ingestion counters gain **Compressed** — samples the door let go — and still reconcile: every sample is stored, compressed, held or lost, exactly once.

Per signal, per group, per connection. In the project, a tag, an acquisition group or a connection can carry `historyCompression`; the most specific one wins, and `"enabled": false` keeps one tag exact under a group that compresses the rest:

```
{
  "name": "Line1",
  "historyCompression": { "deviation": 0.5, "exceptionDeviation": 0.2, "maximumSeconds": 600 },
  "acquisitionGroups": [ { "name": "Fast", "historyCompression": { "deviationPercent": 0.25 } } ],
  "tags": [
    { "name": "Temperature", "acquisitionGroup": "Fast", "writeMinimum": 0, "writeMaximum": 1600
},
    { "name": "BatchCount", "historyCompression": { "enabled": false } }
  ]
}
```

A percentage is taken of the block's own `spanMinimum` / `spanMaximum`, else of the tag's write limits; one with neither is refused when the project is checked. In the Desktop's **I/O configuration**, a tag's pane has a **History compression** section under *Writes*, a group's pane one under its own fields, and a connection's one under *Tag drift*. Each has the same fields — **Deviation** (in engineering units) **or % of span**, **Exception** and **Heartbeat (s)** — and a box to keep the history exactly as acquired (*Keep this tag's history…*, *Keep the history of this group's tags…*, *…of this connection's tags…*). A place whose fields are all blank says nothing of its own and inherits; under its fields it says what from — *Inherited from connection Line1: ±0.5, at least every 600 s*, *Inherited from acquisition group Fast: kept exactly as acquired*, or *Inherited from the historian's rules, where one matches* — and, once it has its own, what they replace. A place's own settings replace the whole of what it would inherit, field by field blank or not. What the project check would say is said in red under that while you type: a percent of span set on a group or a connection is judged against every tag it reaches, so a tag with no write limits is named there (*Tag 'Speed': the deviation is a percent of the span and no span is known…*), and the tag's own pane says the same until it is given limits. A field that is not a number — `0,5` — is kept as typed and refused in words rather than read as another number. In the browser, **Administration ▸ I/O** sets all three levels: a connection has a **History compression** row under its drift settings, and the groups' and the signals' tables a **History compression** column. Each says in a few characters what applies — its own bound (*±0.5*, *±1 %*), *Exact*, or what it inherits (*Inherited (±0.5)*, *Historian's rules*, *Not compressed*), with the sentence saying where from in its tooltip — and opens one dialog: **Inherit**, **Its own settings** (the rule form's fields: deviation or % of span, exception or % of span, span low and high, shortest and longest interval) or **Keep exactly as acquired**. The dialog says where an inherited setting comes from — *Inherited from acquisition group 'fast': ±0.5* — and that nothing is compressed while the historian's switch is off. A percentage of a span is checked before the save against every signal that would take it: a signal with no span of the block's own and no write limits is named beside the box. **Apply** is an edit like any other on that screen; **Save and activate** sends it, and the node checks it again with the project. The historian's own rules (`historian.compression.rules`, first match wins) cover every signal the project does not describe.

What it buys. A day of four signals at one hertz (86,400 readings each), measured on the reconstruction against every reading:

Signal	Deviation	Exception	Kept	Ratio	Largest error
Slow drift (a random walk)	±0.05	—	873	99:1	0.05
Slow drift	±0.2	—	62	1,394:1	0.2
Slow drift	±0.2	±0.5	10	8,640:1	0.5
Noisy measurement (±0.5 of noise)	±0.2	—	57,943	1.5:1	0.2
Noisy measurement	±0.5	—	30,016	2.9:1	0.5
Noisy measurement	±1	—	2	43,200:1	0.97
Setpoint stepping every 40 min	±0.05	—	70	1,234:1	0

Signal	Deviation	Exception	Kept	Ratio	Largest error
Sine with ±0.1 of noise	±0.2	—	3,469	25:1	0.2
Sine with noise	±0.5	—	1,719	50:1	0.5
Sine with noise	±0.2	±0.5	38,004	2.3:1	0.5

Two lessons are in it. A deviation below the noise buys little: the door keeps the noise. And an **exception deviation larger than the compression deviation** turns a smooth signal into a staircase the door then has to store corner by corner — keep it below, about half. The cost is negligible: about 0.2 µs a sample on one core, so 12,000 tags at one hertz take about 0.2 % of one core.

When the store is unavailable

A historian store can go away: a database server restarts, a network link drops, an account's password expires. Acquisition never waits for it.

1. Samples keep being acquired. Live values, alarms, recording to `.opslog` and (if it is a separate store) the embedded historian are unaffected.
2. Samples for the unavailable store are written to the **spool** on local disk. The status bar turns to *Degraded* and shows how many samples are spooled and how full the spool is.
3. When the store comes back, the spool is replayed into it in order and the status returns to *Running*.
4. If the outage outlasts the spool, the **When the spool is full** policy applies. With `StopAccepting` further samples for that store are refused and counted, rather than the oldest being discarded quietly.

A store that is restarting, refusing connections or refusing the login is an outage and is spooled through. A store that accepts the connection but refuses the *data* (a schema that does not match, a value too large for its column) is different: that batch is kept in a **dead-letter file** (`dead-letter.dat`) beside the spool rather than thrown away, and the status line names the file until somebody deals with it.

The default 8 GB spool absorbs roughly:

Acquisition rate	Outage absorbed
100 samples/s	about 30 days
1,000 samples/s	about 3 days
10,000 samples/s	about 7.5 hours
100,000 samples/s	about 45 minutes

Sizing the disk

Order of magnitude, for raw samples before compression, retention and deadband:

Tags	Rate	Raw per day	Raw per year
1,000	1 s	about 2.7 GB	about 1.0 TB
1,000	100 ms	about 27 GB	about 10 TB
10,000	1 s	about 27 GB	about 10 TB
10,000	100 ms	about 273 GB	about 100 TB

Tags	Rate	Raw per day	Raw per year
100,000	1 s	about 273 GB	about 100 TB

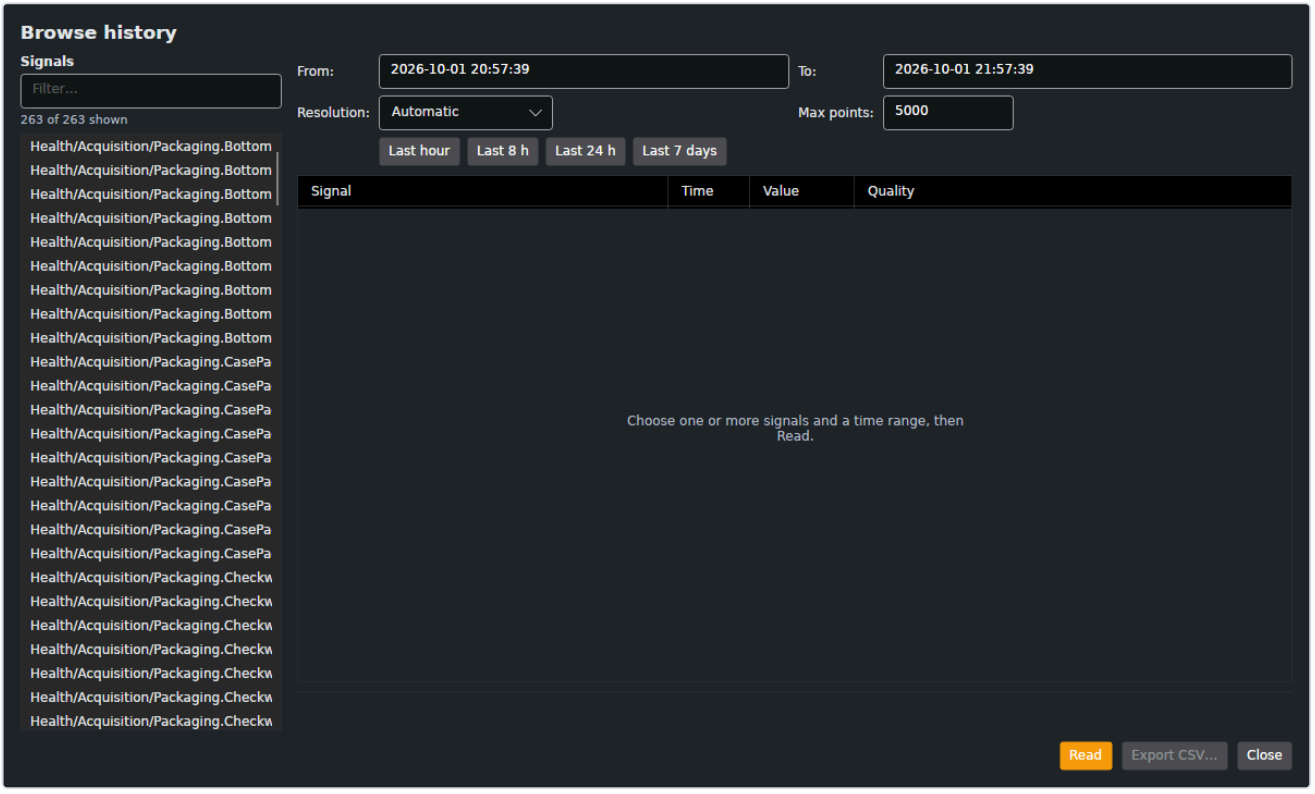
Two things change these figures more than any other:

- **Retention.** Keeping 30 days of raw samples and the rollups after that is a very different curve from keeping raw samples for ever.
- **Deadband.** Most process tags do not change on every scan; a tag with a deadband stores far fewer samples than its poll rate suggests.

To estimate your own node, add up *tags x samples per second* for each acquisition group, multiply by 86,400 seconds and by about 13 bytes (the embedded store's size per sample), and multiply by the retention in days. Add the rollups (small by comparison for signals sampled faster than once a second) and the spool.

Reading history: Data ▸ Browse History

Data ▸ Browse History… reads stored history back as a table: pick signals, pick a window, press **Read**. It asks the same service the data-access API uses, so it selects resolutions, merges stores read alongside, and reports diagnostics exactly as every other reader does. It reads the node's store, and on a client workstation it reads the node's store over the network.



Browse history reading four packaging signals (CaseWeight, WeightTarget, the checkweigher rate and air pressure) over the last hour at one-minute resolution: each row has the bucket value, its quality, the minimum, maximum and number of samples behind it, and the note under the table says which tier served the read

Element	What it does
Signals list	Every signal the historian holds (its catalog). It is multi-select: click with `Ctrl` or `Shift` to choose several. The line beneath says how many are shown and how many are selected, for example *3 of 118 shown · 2 signals selected*, because a selection scrolled out of view is otherwise invisible.

Element	What it does
Filter…	Narrows the list to ids containing the text you type, for example `Checkweigher`.
From: / **To:**	The window, in local time as `yyyy-MM-dd HH:mm:ss`. The window opens on the last hour.
Last hour, **Last 8 h**, **Last 24 h**, **Last 7 days**	Set **From** and **To** to that window ending now.
Resolution:	**Automatic** (the finest tier the point budget buys), **Raw samples** (every stored sample), **1 second** or **1 minute** buckets.
Max points:	The point budget, `5000` by default. It decides the tier under **Automatic**, and it caps how many rows come back.
Bucket value:	What each row is when the answer comes in buckets: **Default** (the mean, weighted by time for a signal that reports on change), **Time-weighted average**, **Mean of the samples**, **Minimum** or **Maximum**. See *What a bucket's value is*. The status line says when values are weighted by time, and when a signal could not be, why.
Result table	**Signal**, **Time** (the header shows the date), **Value**, **Quality** (a green dot and *Good*, or a red dot and the driver's code). A **Unit** column appears when the signals carry units. When rows were served from a rollup tier and summarise more than one sample, **Min**, **Max** and **Samples** columns appear, because such a row is a summary and must not look like a single reading.
Status lines	How many points across how many signals; which resolution was chosen and why (for example "*Chose 1 s for a 1 h window at 5000 points — the finest stored tier that fits.*"); which tier served it and how long the read took. If the budget cut the answer short it says "*cut short by the point budget — narrow the range or raise it*".
Read	Runs the query.
Export CSV…	Saves the rows on screen to a CSV file. Enabled after a successful read.
Close	Closes the window.

If the historian is off or unavailable, the menu item explains why instead of opening the window. If the historian holds no signals yet, the window says "*The historian holds no signals yet.*"

Getting history into trends and other tools

The history browser is a table for checking what was stored. For everything else:

- **Datasets** turn history into reusable extracts (columns, sampling, alignment) that reports and outside tools read by name. See the *Datasets* chapter.
- **Reports** chart and tabulate datasets over a period, including charts of history.
- **Analysis Studio** and run comparison read history to answer *what changed* and *why was this run different*.
- The Desktop's trend graphs show live data and recordings; a trend can be shaded with what the semantic model says the plant was doing (right-click ▸ **Process context**).
- With the Web HMI module, browser trends and history replay read the same historian.

Worked example: retention for the packaging line, then CaseWeight over the last hour

The packaging demonstration acquires 120 tags: ten machine groups polled every 500 ms, `Conveyors` and `CasePacking` every 200 ms and `Line` every second, about **286 samples per second** before deadband, plus computed signals.

Estimate the disk. 286 samples/s x 86,400 s x 13 bytes is about **320 MB of raw history per day**. With 30 days of raw retention that is about **10 GB**, plus the rollups and up to 8 GB of spool. At this rate the 8 GB spool would absorb an outage of well over a week.

Configure the historian.

1. Sign in as an Engineer or Administrator (for example `admin`).
2. Choose **Data ▸ Historian…**.
3. Tick **Keep history in a historian**.
4. On the **Store** tab choose **Provider: Embedded historian**. Leave **Instance id** as `node`, **Data directory** as `Historian` and **Shards** at `16`. Leave **Build tiers finer than the sampling rate** cleared: the Line group is sampled once a second and needs no 100 ms tier.
5. Open the **Ingestion** tab. Leave the queue, batch and flush values at their defaults. Leave **Spool limit (GB)** at `8` and **When the spool is full** at `StopAccepting`.
6. Under **Maintenance** set **Raw retention (days)** to `30`. Leave **Maintenance interval (min)** at `60`, **Context store** at `journal`, and keep **Build rollups for faster long-range queries** ticked.
7. Press **Test**. The message at the bottom should read *Connection succeeded.*
8. Press **Save**. The status bar shows `Historian: embedded`.

The node now keeps 30 days of every raw sample and the 1-minute rollups for as long as the disk allows; a one-year trend of the line rate is served from the 1-minute tier.

Browse CaseWeight over the last hour. Start acquisition (**Acquisition ▸ Start**, `F5`) and let the line run for a while, then:

1. Choose **Data ▸ Browse History…**.
2. Type `Checkweigher` in **Filter…**. The list shows the Checkweigher's nine signals.
3. Click `Packaging/Checkweigher/CaseWeight`, then `Ctrl`-click `Packaging/Checkweigher/WeightTarget`. The count line reads *2 signals selected*.
4. Press **Last hour**. Leave **Resolution** at **Automatic** and **Max points** at `5000`.
5. Press **Read**.

A signal polled every 500 ms produces about 7,200 samples in an hour, more than the budget of 5,000 points per signal, so the selector serves **1-second buckets**: the status line says it chose 1 s for a 1 h window at 5000 points, and **Min**, **Max** and **Samples** columns appear. Each row of `CaseWeight` shows the average case weight in grams over one second, with the lightest and heaviest case in that second beside it.

To see individual readings, set **Resolution** to **Raw samples** and narrow the window, for example **From** to ten minutes ago, then **Read** again. The extra columns disappear, because every row is now one sample. Press **Export CSV…** to save the rows, for example as `caseweight-last-10-min.csv`.

Moving history between stores

Sooner or later a site wants history somewhere it was not written: the embedded store's history in a SQLite file or a server database, or ten years of an incumbent historian's data beside what Meelmo has acquired since. There are three tools for it, and they combine:

Tool	Copies anything?	Where	Use it to
Read alongside	No	**Data ▸ Historian…**, **Read alongside** tab	Keep an old store readable, so a trend across the change-over shows both halves as one line. Nothing listed there is ever written to.

Tool	Copies anything?	Where	Use it to
Also write to a second store, for a cutover	Only new samples	**Data ▸ Historian…**, **Migration** tab	Send everything acquired from now on to the new store as well, while the older history is copied.
`Meelmo.Migrate`	Yes, the history already held	A command-line program	Copy history from one store, or from an export file, into another, oldest first (or newest first with `--newest-first`), resumably.
Copy existing history into the new store	Yes, what the old store held before the switch	**Data ▸ Historian…**, **Store** tab; **Administration ▸ Historian** in the browser	Switch the node to a new store and have the old store's history follow it, with nothing else to run.

Copying history when you switch stores

Switching the node's historian — from the embedded store to SQLite, say — opens a store that is empty. Acquisition carries straight on into it and nothing is lost, but every trend that reaches back past the switch reads the new store, so to the person looking at it, everything before the switch has disappeared. The old store still holds it.

Tick **Copy existing history into the new store** when you save (on the Desktop's **Store** tab, or in the browser's **Administration ▸ Historian** under the store picker), and the node copies it across:

- **After the switch, behind acquisition.** The node opens the new store first, exactly as without the copy; acquisition and live recording go to it at once and are never paused. The copy then reads the old store and writes into the new one in the background, newest first: the last six hours of every signal before the switch, then the six hours before those, and so on back to the beginning.
- **Only what came before the switch.** It copies up to the instant the new store began taking samples, and no further, so nothing the new store recorded itself is written twice.
- **Switching back copies only what was missed.** A store the node used before already holds its own history up to the moment it was left. Switched back to it, the copy starts from that moment — the last stop in the store's own record of its sessions — so a store that would keep a second copy of a sample (the embedded one) does not get its history twice. A store that recognises a sample it already holds (SQLite, TimescaleDB, QuestDB, ClickHouse) takes the whole of the old store's history again, and keeps one of each.
- **Held to a rate.** At most 50,000 samples a second, so it never competes with the plant's own samples for the store. On an ordinary four-core server, three million samples (a hundred signals, a little over eight hours at one a second) copied from the embedded store into SQLite took 20 seconds flat out — about 150,000 a second — and 61 seconds at the ceiling.
- **Resumable.** The job is written to `history-copy.json` in the node's historian folder, with its progress in `history-copy.checkpoint.json` beside it. If the node stops in the middle, the copy carries on from the checkpoint when the node next opens that store. A finished or cancelled copy removes both files.
- **Visible.** The Desktop's **Migration** tab and the browser's **Copying history** panel show the share done, the samples copied, the rate, the time left and anything that could not be copied, with **Cancel copy**. While the copy runs they also say how far back the new store is already whole — *everything from 28/10/2026 12:00 up to the switch* — so you know which trends read complete; that line is withdrawn if a window fails, since the hole it leaves would make it untrue. The node's log says when it starts, how it is doing every half minute, and the migrator's own summary when it ends.
- **The old store is only read, and kept.** Nothing is written to it and nothing deletes it. Once the copy has finished and the trends look right, deleting the old store's folder or file is your decision.

Trends read the new store while the copy runs, so they fill in as it goes: what is already copied is there, and the rest is drawn as a gap until the copy reaches it. Because the copy goes newest first, the hours just before the switch — what the trends people open first show — are the first to arrive, and the years behind them fill in backwards. The old store is not read alongside meanwhile. Reading both would count the copied stretch twice in every average and total over it, and that stretch is the one that keeps growing. To keep the old store readable for longer — or instead of copying — add it on **Read alongside**.

If the copy cannot finish — the old store will not open, or the new one turns the writes away (a SQLite file whose schema was never created is the usual reason) — it stops, says why, and keeps its checkpoint. It is not retried by itself; fix the cause and press **Retry**, or **Dismiss** it. Switching the node to yet another store stops a copy into the store being left, for the same reason.

The copy is offered only where the node runs in the same process as the screen: the Desktop running its own node, or the web host running one. A screen attached to a node on another machine does not offer it, because the old store's files are on that machine; use `Meelmo.Migrate` there.

What Meelmo.Migrate is

`Meelmo.Migrate` is a separate program in the `tools` folder of the installation (`Meelmo.Migrate.exe` on Windows). It is not a mode of the node or of the data-access server, for three reasons: it writes to a store the server only reads, it runs for hours rather than for one request, and you must be able to start it, stop it and resume it without touching the service the plant depends on.

It reads from one of three kinds of source and writes into any historian store this machine is licensed for:

Source kind (`--source-kind`)	`--source` names	What is read
`provider`	A configuration file (JSON)	A store Meelmo wrote: the embedded store, a SQLite file, or a SQL Server, Oracle, TimescaleDB, QuestDB or ClickHouse historian.
`existing-schema` (or `existing`)	A configuration file (JSON)	An incumbent historian's own database tables, read in place and never written to. The configuration must say `"accessMode": "ExistingSchemaReadOnly"` and map the tables.
`csv`	A CSV file, or a folder of them	An incumbent's CSV export, PI's archive export among them. Read once from start to end and never written to.

The target is always a configuration file. The embedded store and SQLite are in the base product; a SQL Server, Oracle, TimescaleDB, QuestDB or ClickHouse target needs its historian module. The tool checks the licence the way the rest of the product does, so it cannot be used to reach a store the licence does not cover.

> **The embedded store is held by one process at a time.** While a node writes to an embedded store, nothing else can open it, `Meelmo.Migrate` included. To copy history out of the node's embedded store, or into one, stop the node first: stop the recorder service, or close the Desktop that hosts the node. SQLite and the server databases can be read and written while the node runs.

The configuration files

`--source` (for a store) and `--target` name JSON files that describe a store: its provider, its instance id, and the provider's settings. Write them by hand. The file **Export historian.json…** saves is meant for the data-access server and the web host; it starts with comment lines, which this tool does not read.

Key	Meaning
`providerId`	Required. `embedded`, `sqlite`, `sqlserver`, `oracle`, `timescale`, `questdb` or `clickhouse`.
`instanceId`	The store's instance id. It appears in provenance and in the checkpoint, so two stores holding different data must not share one.
`settings`	The provider's own settings, by key. For `embedded`: `dataDirectory`, `shardCount`, `buildSparseTiers`. For `sqlite`: `filePath`, `batchSize`, `commandTimeoutSeconds`. The server providers' keys are given in each server historian's chapter.
`secretRefs`	Passwords, as sealed tokens. A sealed token opens only on the machine that sealed it, so run the tool on the machine that holds the historian configuration.
`accessMode`	`ReadWrite` (the default) or `ExistingSchemaReadOnly` for an incumbent's tables.
`mapping`	With `ExistingSchemaReadOnly` only: where the data is in the incumbent's tables. The same fields as the **Historical Source** window's *A database this product did not create* (see *Read alongside tab* above).

Write every path in full. A file store that is not there yet is created by a real run, never by a dry run.

```
{
  "providerId": "embedded",
  "instanceId": "node",
  "settings": { "dataDirectory": "D:\\Meelmo\\Historian" }
}
```

```
{
  "providerId": "sqlite",
  "instanceId": "packaging-sqlite",
  "settings": { "filePath": "D:\\Meelmo\\packaging-history.db" }
}
```

Every run is profiled before it writes

Before anything is written, every run reads both sides and prints an **import profile**. A dry run (`--dry-run`) is the same profile with nothing after it, so a dry run and a real run cannot disagree about whether an import is safe to start. Building the profile writes to neither store.

The profile states the source and the target, how many signals the selection matched and how many hold data, the number of samples and their time range, the **read rate** measured by reading the first window of the source on this machine, the **ceiling** if you set `--max-rate`, and how long reading will take at least. Writing is not measured by a dry run, so that time is a floor. If the first window alone filled one read, the profile says the windows will be split, which is the first sign that `--window` is too wide for this history.

Then it lists its **findings**. A *note* is worth knowing and does not stop the run. A finding marked **REFUSES** stops the run before the first write, until you fix the cause or accept the finding by its code with `--accept <code>`, having read it. `--accept` is repeatable and names one code at a time on purpose: a switch that accepted every refusal would be the switch everybody learnt to type. Accepted findings are shown as *accepted*, and they are written into the import report and the audit trail.

Code	Kind	What it means and what to do
`overlaps-target`	Refuses	Signals would land inside a period the target already holds, usually the months the new node has been acquiring, and two systems' values for one signal would be interleaved. Import up to the first instant the target holds with `--to`, or accept it if this is the window an interrupted run of the same import was writing.

Code	Kind	What it means and what to do
`type-conflict`	Refuses	A signal is text on one side and numbers on the other. Rename one side, or exclude the signal.
`nothing-selected`	Refuses	The selection matched no signal that holds anything. Check `--include`, `--exclude`, `--from` and `--to`.
`future-samples`	Refuses	Samples are stamped more than five minutes after now. That is almost always a time zone or a clock: an export written in local time and read as UTC.
`unreadable-rows`	Refuses (CSV)	Rows whose timestamp or tag could not be read, and that would be left out. Fix the export, give the format with `--csv-time-format`, or accept that they are not imported.
`ambiguous-local-time`	Refuses (CSV)	Rows stamped in the hour that happens twice when the clocks go back in the `--csv-time-zone` zone. They would be placed in the second of the two hours. Export in UTC, or accept it.
`already-imported`	Note	The checkpoint says every selected signal has already been moved. There is nothing left to do.
`resuming`	Note	Resuming from the checkpoint: the signals already done are skipped, and every other one starts after the last instant the target confirmed.
`no-checkpoint`	Note	The target does not recognise a re-sent sample and no `--checkpoint` is set.
`system-states`	Note (CSV)	Values that are PI's word for having no reading (`Shutdown`, `I/O Timeout`, `Bad Input` and the like). They are not stored, and the report counts them as *no value*.
`out-of-order`	Note (CSV)	Readings earlier than the one before them for the same signal. They are imported where they belong in time.
`outside-range`	Note (CSV)	Rows outside `--from` and `--to`, not imported.

The profile ends with *The import may run.* or with the list of codes still refusing.

Options

`Meelmo.Migrate --help` prints the same list. An option the tool does not know, or a value it cannot read, is an error rather than something ignored: a run told `--windwo 1:00:00` stops at once instead of running for hours with the default.

Required

Option	Default	What it does
`--source <file>`	—	The source. A configuration file for a store; with `--source-kind csv`, the CSV file or a folder of `*.csv` files, read in name order.
`--target <file>`	—	The target store's configuration file.

Common

Option	Default	What it does
`--source-kind <kind>`	What the source configuration declares	`provider`, `existing-schema` (also `existing`) or `csv`. Given for a store, it must agree with the configuration: `provider` over a configuration that declares `ExistingSchemaReadOnly`, or `existing-schema` over one that does not, is refused before anything is opened.
`--checkpoint <file>`	None	The resume state. Strongly recommended: re-running with the same checkpoint retries only what did not move. For a store it records, per signal, the last instant the target confirmed; for a CSV export, the row reached in each file. A CSV file that has changed since its checkpoint was written is refused rather than resumed.
`--dry-run`	Off	Profile the import and exit, writing nothing: not the schema, not a missing SQLite file, not an embedded store's folder. Exit code `0` when a run may start, `6` when it would be refused.
`--plan`	Off	The old name of `--dry-run`. Does the same.
`--accept <code>`	None	Accept one refusing finding by its code. Repeatable.
`--report <file>`	`<state>/migrations/import-<yyyyMMdd-HHmmss>.txt`	Where to keep the import report. The report is printed either way and always written to a file; this only chooses the file, for example inside the commissioning pack.
`--create-schema`	Off	Create the target's tables before migrating. A file store has no database administrator, so you say it here; with a server database an administrator normally creates the schema (see that provider's chapter). Ignored by a dry run, which profiles a target with no schema yet as an empty one.
`--window <duration>`	`6:00:00`	The time span read and written as one unit. A window whose read is cut short by `--max-points` is split in half until it fits, down to one second.
`--from <instant>`	The earliest held	Start of the period, for example `2025-01-01T00:00:00Z`. A time without an offset is read as UTC.
`--to <instant>`	The latest held	End of the period. Must be after `--from`.
`--include <prefix>`	Everything	Copy only signals under this path, for example `Packaging/Checkweigher`. Repeatable. A prefix matches whole path levels, exactly as written (upper and lower case differ), and has no trailing `/`: `Packaging` matches `Packaging/Checkweigher/CaseWeight`, `Packaging/` matches nothing.
`--exclude <prefix>`	Nothing	Leave out signals under this path. Repeatable. Exclusion wins over inclusion.

Not disturbing the node

Option	Default	What it does
`--max-rate <n>`	No ceiling (`0`)	At most *n* samples written per second. Use it when the target is a store a node is acquiring into, so the node's own samples keep the store's attention.

Option	Default	What it does
`--priority <p>`	`low`	`low` runs the tool below normal priority, so the acquisition sharing the machine keeps the processor; `normal` does not. A platform that will not lower the priority does not stop the run.

Speed

Option	Default	What it does
`--signals-per-read <n>`	`25`	Signals read together in one window.
`--no-read-ahead`	Read ahead	Read each window only after the previous one is written. By default the next window is read while the current one is written.
`--newest-first`	Oldest first	Copy the most recent history first and work back: every signal's last window before any signal's earlier ones, so the recent past is readable in the target early. A checkpoint resumes a run in either direction, and a run in the other direction copies only what is still missing.

CSV exports (only with `--source-kind csv`; with any other kind they are an error)

Option	Default	What it does
`--csv-prefix <path>`	None	The path every tag lands under, for example `Incumbent/PI`. A tag `FT101` becomes the signal `Incumbent/PI/FT101`.
`--csv-time-zone <id>`	`UTC`	The zone a timestamp without an offset was written in, as an IANA name such as `Europe/Berlin`. A zone this machine does not know is an error. A timestamp that carries its own offset (`Z`, `+01:00`) keeps it.
`--csv-time-format <format>`	The recognised formats	An exact .NET timestamp format, for an export none of the recognised formats reads, for example `dd.MM.yyyy HH:mm:ss`.

Less common

Option	Default	What it does
`--no-validate`	Validate	Skip the comparison made after each window is written. By default, once a window is written, the statistics of each signal over that window are asked of both stores and compared, and the checkpoint advances only after the write and the comparison succeed.
`--stop-on-failure`	Carry on	Stop at the first failed window. By default the run carries on, and the report names the windows that failed.
`--max-points <n>`	`250000`	The most points read for one signal in one read.
`--batch <n>`	`5000`	Samples per write to the target.
`--allow-rerun-without-dedup`	Off	Run without `--checkpoint` into a target that does not recognise a re-sent sample. Reasonable once: a first run into an empty store. Without it, such a run is refused with a configuration error.
`--state <dir>`	The `Meelmo` folder under your local application data	Where the tool finds the licence and entitlements, and where it writes its audit trail (`<state>/audit`) and reports (`<state>/migrations`).

Option	Default	What it does
`--verbose`	Off	Debug logging.

Which targets recognise a re-sent sample: **SQLite**, **TimescaleDB**, **QuestDB** and **ClickHouse** do, so a re-run over windows already copied stores nothing twice (ClickHouse keeps the second copy on disk until its next merge, and counts it nowhere). The **embedded** store, **SQL Server** and **Oracle** do not, which is why a checkpoint matters most for them.

CSV exports

The first line of every file must be a header. Two shapes are read, chosen from the header:

Shape	Header	Typical source
One row per reading	A tag column (`tag` , `tagname` , `tag name` , `name` , `signal` , `point` , `pointname` , `point name`), a time column (`timestamp` , `time` , `datetime` , `date time` , `date` , `time stamp` , `ts` , `_time`), a value column (`value` , `val` , `pv` , `_value`), and optionally a quality column (`status` , `quality` , `q` , `flags`)	PI's tag and archive exports
One row per instant	A time column first, then one column per tag, named in the header	Spreadsheet exports, PI DataLink

Column names are matched without regard to case. The separator is a comma, a semicolon or a tab, whichever the header uses most, and fields may be in double quotes.

- **Values.** A number is stored as a number. Anything else is kept as text, except one of PI's system states, which is counted as *no value* and not stored.
- **Quality.** An empty quality is good. A number is taken as the quality code. `good` or `ok` is good, `bad` or `invalid` is bad, and any other word is uncertain.
- **Timestamps.** A timestamp with an offset is read as written. Otherwise these forms are recognised, with or without fractions of a second: `yyyy-MM-dd HH:mm:ss` (also with `T` , or without seconds), `yyyy/MM/dd HH:mm:ss` , `dd-MMM-yy HH:mm:ss` and `dd-MMM-yyyy HH:mm:ss` (PI's style, `14-Mar-25 06:00:00`), and `M/d/yyyy h:mm:ss tt` , `M/d/yyyy H:mm:ss` and `M/d/yyyy H:mm` . A local time that never happened (the hour skipped when the clocks go forward) is an unreadable row.

Exit codes

Code	Meaning
`0`	Success. With `--dry-run` : the import may run. `--help` also exits with `0` .
`1`	Cancelled. `Ctrl+C` stops after the current window, with progress checkpointed.
`2`	A configuration problem the operator can fix: an unknown option, a missing file, a source kind that disagrees with its configuration, a target whose schema is not ready, a run into a store that does not recognise a re-sent sample without a checkpoint. The message says which. Starting the tool with no options at all prints the usage and exits with `2` .
`3`	An unexpected error, printed in full. An embedded store held by a running node ends here.
`4`	Finished, but some windows failed. The report names them; run again with the same checkpoint.
`5`	Finished, but the audit trail could not record it. The samples are in the target; keep the printed report with the migration's paperwork.
`6`	Refused on the profile's evidence before anything was written. With `--dry-run` : a run would be refused.

The report and the audit trail

Every run that moves data writes two entries to the audit trail under `<state>/audit`: one before the first write (`historian.import.started`), naming the source and the target and any refusals accepted, and one after the last (`historian.import`), saying what moved. So an interrupted migration still says which store was read and which was written. If the first entry cannot be written, the run does not start; make the folder writable, or point `--state` somewhere that is.

At the end the tool prints the **import report**: the profile, the refusals accepted, the counts, how the two sides were compared, and the verdict. It writes the same text to the `--report` file, or to `<state>/migrations/import-<timestamp>.txt`, and says where.

```
$ Meelmo.Migrate --source riverside-pi-archive.csv --source-kind csv --target sqlite.json --csv-prefix Incumbent/PI -
-csv-time-zone Europe/London --checkpoint riverside.ckpt --create-schema --dry-run
The target /tmp/meelmo-webshots-yBq4aL/backfill/riverside-history.db does not exist yet. A run with --create-schema cr
eates it; a dry run creates nothing, so the import is profiled against an empty target.
Import profile (read only — nothing was written)
  source          csv:/tmp/meelmo-webshots-yBq4aL/backfill/riverside-pi-archive.csv
  target          sqlite
  signals         4 selected, 4 with data
  samples         8,642
  range           2025-10-24T23:00:00.0000000+00:00 .. 2027-10-27T08:16:00.0000000+00:00
  read rate       101,282 samples/s, measured reading the source on this machine
  reading takes   at least 0.1 s (writing is not measured by a dry run)

  REFUSES [unreadable-rows] 1 row(s) could not be read and would be left out. Fix the export, give the timestamp form
at, or accept that they are not imported. For example: riverside-pi-archive.csv line 8,644: '26-Oct-25 1' is not a tim
estamp this import reads (give its format with --csv-time-format)
  REFUSES [ambiguous-local-time] 240 row(s) are stamped in the hour that happens twice when the clocks go back in Eur
ope/London, so which of the two hours each belongs to cannot be known; they would be placed in the second. Export in U
TC, or accept it. For example: riverside-pi-archive.csv line 752 (2025-10-26T01:00:00.0000000+00:00 as read)
  REFUSES [future-samples] 1 signal(s) hold samples stamped after now. That is almost always a time zone or a clock:
an export written in local time and read as UTC lands hours from where it happened. For example: Incumbent/PI/RIV.CLA
R.AT201.PV reaches 2027-10-27T08:16:00.0000000+00:00.
  note    [out-of-order] 4 reading(s) are earlier than the one before them for the same signal. They are imported whe
re they belong in time; a resumed import is still exact, because it resumes by row.

The import will not run until each refusal above is fixed, or accepted with --accept <code> by somebody who has read i
t: unreadable-rows, ambiguous-local-time, future-samples.

Nothing was written, and a run would be refused on the evidence above.
```

Meelmo.Migrate profiling a CSV export of the Riverside works' PI archive into an empty SQLite store: four signals and 8,642 samples, the read rate, and three refusals — a row cut short by the export, 240 rows stamped in the hour that happens twice when the clocks go back in Europe/London, and an analyser whose samples reach a year ahead — with a note on readings out of order; nothing was written

The order for a cutover to a new store

1. On the **Migration** tab, tick **Also write to a second store, for a cutover**, choose the new store under **Second store:**, and press **Save**. Everything acquired from now on goes to both stores.
2. Run `Meelmo.Migrate --dry-run` from the old store into the new one, with `--to` set to the moment you saved in step 1, so the copy does not overlap what the second store has received since. Read the profile.
3. Run it again without `--dry-run`, with a `--checkpoint`. Re-run with the same checkpoint until the report shows no failed windows.
4. Check that the second store reports nothing lost.
5. On the **Store** tab, make the new store the historian and clear **Also write to a second store, for a cutover**. To keep reading the old store rather than delete it, add it on **Read alongside** as *A Meelmo store kept after a migration*. Press **Save**.

If the old store is an embedded store, steps 2 and 3 need the node stopped (see the note above). A short stop is usually easier the other way round: switch the node to the new store first, then copy the old embedded store, which no process holds any more, as in the first worked example below.

Worked example: the packaging line's embedded history into SQLite

The packaging demonstration has kept its history in the embedded store, in `D:\Meelmo\Historian`. The site wants it in one SQLite file that its own tools can read.

1. **Switch the node to the new store.** Choose **Data ▸ Historian…**. On the **Store** tab choose **Provider: SQLite**, set **Instance id** to `packaging-sqlite` and **Database file** to `D:\Meelmo\packaging-history.db`. Press **Test**, then **Create schema** when it appears, then **Save**. From now on new samples go to SQLite, and no process holds the embedded store.

2. **Write the two configuration files** shown under *The configuration files* above, as `D:\Meelmo\migrate\embedded.json` and `D:\Meelmo\migrate\sqlite.json`.

3. **Look first.** In a command prompt, in the installation's `tools` folder:

   ```
   Meelmo.Migrate --source D:\Meelmo\migrate\embedded.json ^
                  --target D:\Meelmo\migrate\sqlite.json --dry-run
   ```

   The profile lists the packaging signals and their samples, the time range, and the read rate measured on this machine. The embedded store's history ends where the node's writing to SQLite began, so nothing overlaps; had the two periods overlapped, the profile would refuse with `overlaps-target`. SQLite recognises a re-sent sample, so there is no `no-checkpoint` note. It ends with *Nothing was written. Remove **--dry-run** to run it.* and exit code `0`.

4. **Run it.** The same command, without `--dry-run` and with a checkpoint:

   ```
   Meelmo.Migrate --source D:\Meelmo\migrate\embedded.json ^
                  --target D:\Meelmo\migrate\sqlite.json ^
                  --checkpoint D:\Meelmo\migrate\packaging.ckpt
   ```

   A progress line every two seconds gives the share done, the samples written, the rate and the window being copied. At the end the report is printed and its file named, under `migrations` in your local application data.

5. **If it is interrupted**, press nothing else: run the same command again. The profile notes `resuming` and the run carries on where the target last confirmed.

6. **Check. Data ▸ Browse History…**, `Packaging/Checkweigher/CaseWeight`, a window from before the switch: the rows now come from SQLite.

Because SQLite recognises a re-sent sample, a re-run over windows already copied stores nothing twice. Into an embedded, SQL Server or Oracle target, the checkpoint is what prevents that.

Worked example: an incumbent's PI export beside the packaging line

Before Meelmo, the packaging line was historized by PI. The site exported a year of the checkweigher's tags from PI's archive as CSV files into `D:\pi-export`, one row per reading, with timestamps in local time (Europe/Berlin):

```
Tag,Timestamp,Value,Status
PACK_CW_WEIGHT,14-Mar-25 06:00:00,12.46,Good
PACK_CW_WEIGHT,14-Mar-25 06:00:01,12.49,Good
PACK_CW_REJECTS,14-Mar-25 06:00:00,Shutdown,
```

The target is the node's SQLite store from the previous example, which the node is writing to.

1. **Look first**, landing the tags under their own path so they can never collide with the line's signals:

   ```
   Meelmo.Migrate --source D:\pi-export --source-kind csv ^
                  --csv-prefix Incumbent/PI --csv-time-zone Europe/Berlin ^
                  --target D:\Meelmo\migrate\sqlite.json --dry-run
   ```

   `PACK_CW_WEIGHT` lands as `Incumbent/PI/PACK_CW_WEIGHT`. The `Shutdown` rows are counted under the `system-states` note, not stored. Because the year spans the last Sunday of October, the profile refuses with `ambiguous-local-time`: the rows stamped between 02:00 and 03:00 that night cannot be placed in the first or the second of the two hours. Exit code `6`.

2. **Decide.** Ask for the export again in UTC, which removes the refusal, or read the finding and accept that those rows are placed in the second hour.

3. **Run it**, here accepting the finding, with a checkpoint and a ceiling so the node's own writing keeps the store's attention:

   ```
   Meelmo.Migrate --source D:\pi-export --source-kind csv ^
                  --csv-prefix Incumbent/PI --csv-time-zone Europe/Berlin ^
                  --target D:\Meelmo\migrate\sqlite.json ^
                  --checkpoint D:\Meelmo\migrate\pi-export.ckpt ^
                  --max-rate 20000 --accept ambiguous-local-time
   ```

   The accepted code is written into the audit trail's first entry and into the report.

4. **Use it.** The incumbent's history is now in the same store as the line's, under `Incumbent/PI`, and every history reader (Browse History, datasets, reports, analysis) can put last year's `PACK_CW_WEIGHT` beside this year's `Packaging/Checkweigher/CaseWeight`.

To read an incumbent's database instead of an export, leave it where it is and either add it under **Read alongside** as *A database this product did not create*, or copy it with `--source-kind existing-schema` and a configuration that declares `"accessMode": "ExistingSchemaReadOnly"` and its `mapping`. Either way the incumbent is only read.

The semantic model

The historian stores signals: numbers, booleans and text, each with a time. A person asks questions in other words: *how long did order 4711 take*, *how often did the checkweigher fault this week*, *what was the average case weight while the line was producing*. The **semantic model** is what connects the two. It says what the plant is made of (assets), what each signal means to its asset (roles), and what the signals' behaviour means (runs, events and states found by detection rules).

This chapter covers the four Desktop windows that describe the semantic model: **Data ▸ Asset model…**, **Data ▸ Role catalogue…**, **Acquisition ▸ Detection Rules…**, and **Data ▸ Export Semantic Model…** / **Import Semantic Model…**. It ends with how the model is used by trends, datasets, reports, analysis and the assistant, and a worked example on the packaging line.

The primitives

The engine has seven generic primitives. It knows nothing about pumps, fillers or checkweighers; your words for things live in your project and are matched, never interpreted. That is what lets the same product serve a water treatment works and a packaging line.

Primitive	Is	Packaging line example
Asset	A thing in the plant: a machine, a line, an area	`Packaging/Checkweigher`
Signal	A measurement an asset produces	`Packaging/Checkweigher/CaseWeight`
Event	Something that happened at an instant, or over a short interval	*Checkweigher fault*
Execution	Something that ran over a period: a batch, an order, a wash cycle	*Production order*
State	A condition that held over a period	*Line mode*: Producing or Changeover
Context	A named value that applied over a period	the SKU of the order being run
Relationship	A link between assets: feeds, contains, is upstream of	the Loader feeds the Checkweigher

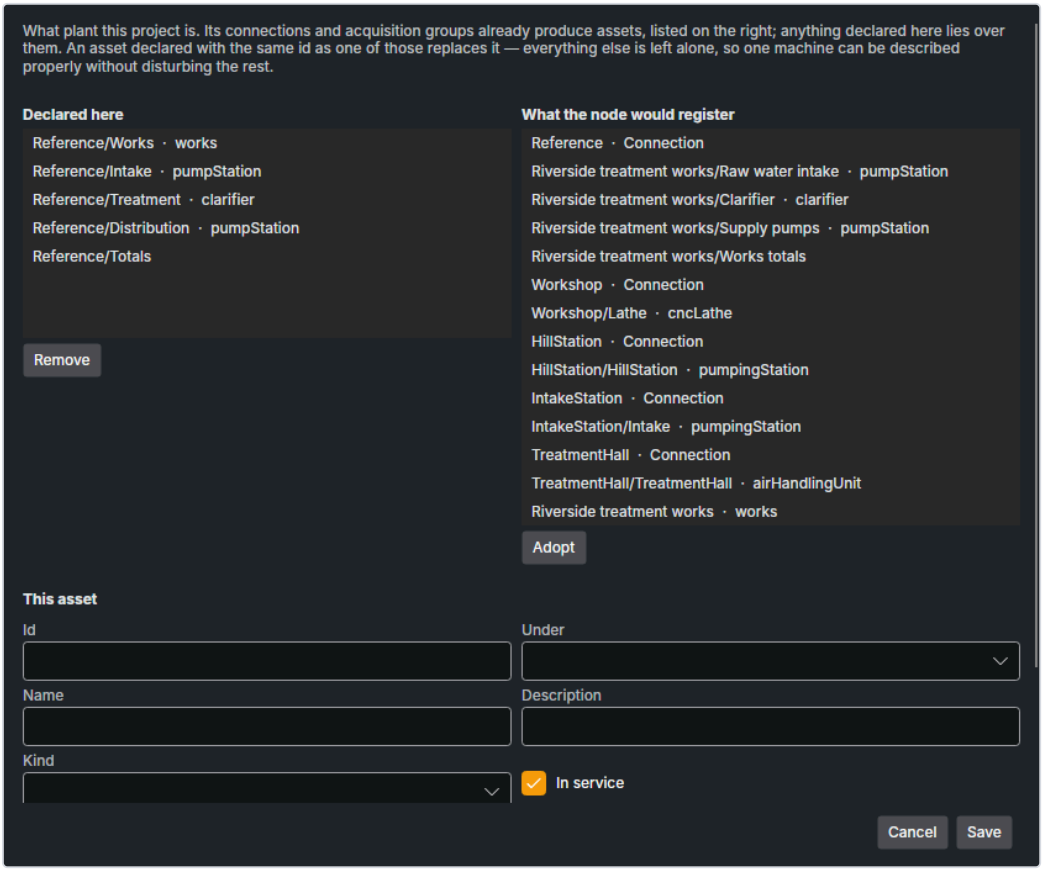
Where each part of the model is kept matters when you move or restore a plant:

Part	Edited in	Stored in
Asset model	**Data ▸ Asset model…**	The project (travels in the `.meelmoproj`, part of each configuration revision)
Role catalogue	**Data ▸ Role catalogue…**	The project
Kind of asset of a group, Role of a tag	**Acquisition ▸ I/O Configuration…**	The project
Detection rules	**Acquisition ▸ Detection Rules…**	The node (the rules file named on the historian's **Read alongside** tab), and the project for the demonstration plants
Detected runs, events and states	Nothing: the node writes them	The historian's context store (**Context store** on the **Ingestion** tab)

What plant this is: the asset model

Data ▸ Asset model… says which things the project contains. You do not start from a blank page: the project already implies assets. Every enabled **connection** is an asset of kind `Connection`, and every **acquisition group** under it is an asset too, of kind `Area` unless the group says otherwise. On the packaging line that gives `Packaging` and one asset per group: `Packaging/Erector`, `Packaging/BottomSealer`, `Packaging/Loader`, `Packaging/Checkweigher` and so on to `Packaging/CasePacking`.

That derivation describes the shape of the *acquisition*, and a plant is not always laid out to suit a PLC: a filler fed by two controllers is two groups but one asset, and a group named for its poll rate becomes an asset called `r500ms`. What you declare in this window **lies over** the derived assets. Declare an asset with the same id as a derived one and it replaces it; everything else is untouched. So one machine can be described properly without disturbing the other forty.



The Asset model window on the packaging line: nothing declared yet on the left, the fourteen assets the node would register on the right (the Packaging connection and one Area per acquisition group), and the form for one asset below

The window

Element	What it does
Introduction	"*What plant this project is. Its connections and acquisition groups already produce assets, listed on the right; anything declared here lies over them…*"
Declared here	The assets this project declares. Select one to load it into the form. An asset that is not in service is marked *not in service*.
Remove	Removes the selected declared asset. If it replaced a derived asset, the derived one comes back.
What the node would register	The assets the node registers when this project is activated: derived and declared together, each as `id · Kind`.

Element	What it does
Adopt	Copies the asset selected on the right into **Declared here** with its id already filled in. This is how you *replace* a derived asset rather than add a second one beside it.
This asset	The form for the selected or new asset (fields below).
Add	Declares a new asset from the form.
Apply	Writes the form back into the selected declared asset.
Save / **Cancel**	Saves the asset model into the project, or closes without changing it. Save the project afterwards (**File ▸ Save**) and activate it to reach the node.

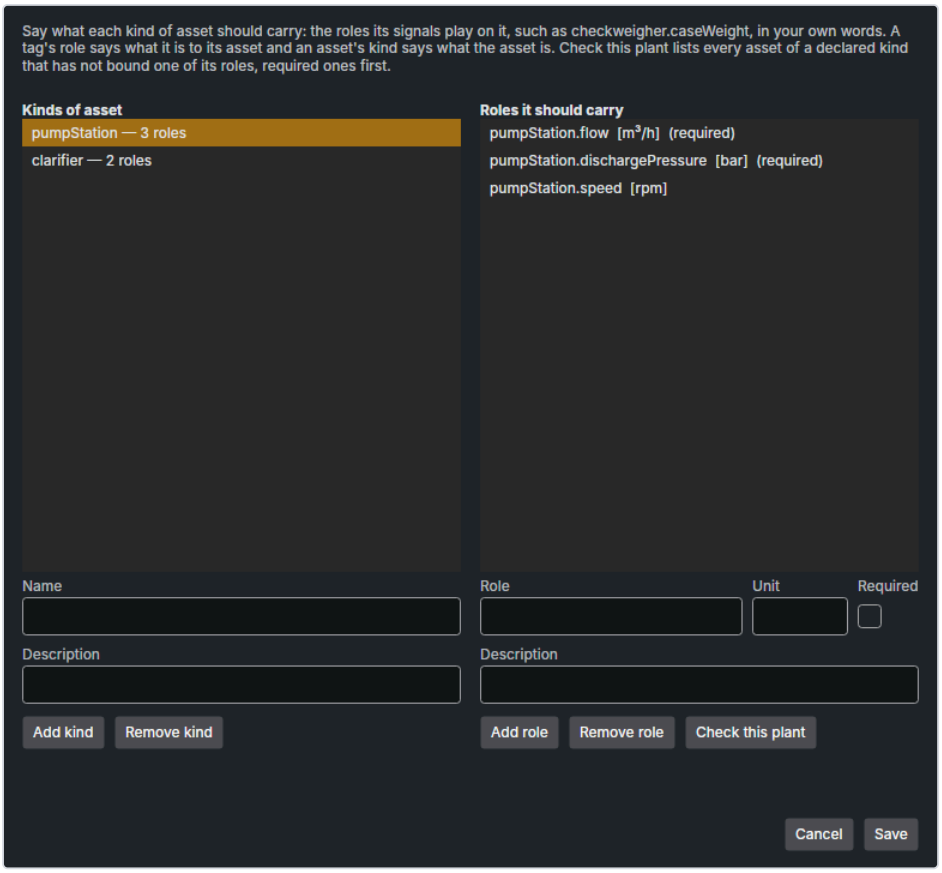
Field	Meaning
Id	What signals, datasets and reports name this asset by, for example `Packaging/Checkweigher`. It cannot be changed once declared. To replace a derived asset, adopt it rather than typing the id.
Under	The parent asset, chosen from the list of known ids. The asset's **path**, which pages, datasets and reports scope by, is built from this chain rather than typed, so it cannot disagree with the hierarchy.
Name	The readable name, for example *Checkweigher*.
Description	Free text for the people who read the model.
Kind	What kind of thing this is. The list offers the kinds declared in the role catalogue; you may also type one. The kind is what the role catalogue's check matches against.
In service	Cleared for an asset that exists but is not currently in use.

The note under the form repeats the one rule that catches people out: "*The id is what signals, datasets and reports name this asset by, so it cannot be changed once declared. To replace an asset the project already derives, select it on the right and adopt it.*"

What each kind of asset is made of: the role catalogue

Data ▸ Role catalogue… says what each **kind** of asset should carry. A checkweigher carries a case weight and a weight target, and some of those are required. Nothing in the product knows what a checkweigher is; the words are entirely yours.

The catalogue is the *type* side of the model. A tag's **Role** says what that tag is to its asset (the case weight), a group's or asset's **Kind** says what the asset is (a checkweigher), and the catalogue says what that kind ought to have. Together they let a dataset or report be written for *a kind of machine* and pointed at any one of them, instead of being a report about one machine for ever.



The Role catalogue window: the kinds of asset on the left, the roles the selected kind should carry on the right, with the fields to add each and the Check this plant button

The window's title bar reads **Role catalogue**. The line at the top says what it is for: the roles each kind of asset's signals should play, and that **Check this plant** lists the assets of a declared kind that have not bound one of them yet. Every kind and role in the catalogue is yours to edit.

Element	What it does
Kinds of asset	The kinds this project declares. Select one to see and edit its roles.
Name / **Description** *(left)*	The kind's name, for example `Checkweigher`, and a description.
Add kind / **Remove kind**	Adds the kind in the fields, or removes the selected one.
Roles it should carry	The roles of the selected kind. A required role is shown with *(required)*.
Role	The role's name, for example `checkweigher.caseWeight`. This is the text a tag's **Role** box must contain.
Unit	The unit the role is expected in, for example `g`.
Required	Ticked when every asset of this kind must bind this role.
Description *(right)*	What the role means.
Add role / **Remove role**	Adds the role in the fields to the selected kind, or removes the selected role.

Element	What it does
Check this plant	Checks every asset of every declared kind against the tags of this project and lists which instance has not bound which role, required ones first (up to twelve lines, then *…and N more*). When everything is bound it says "*Every declared role is bound on every instance.*"
Save / **Cancel**	Saves the catalogue into the project, or closes without changing it.

The check works on the project, with nothing running, because a plant is usually described before it runs. It only examines assets that **have a kind**: an asset whose kind the catalogue does not declare (including every group left as `Area`) is skipped, not reported, because an undeclared kind is not an incomplete one.

Assigning kinds and roles to signals

The binding itself is done in **Acquisition ▸ I/O Configuration…**, beside the rest of each group and tag:

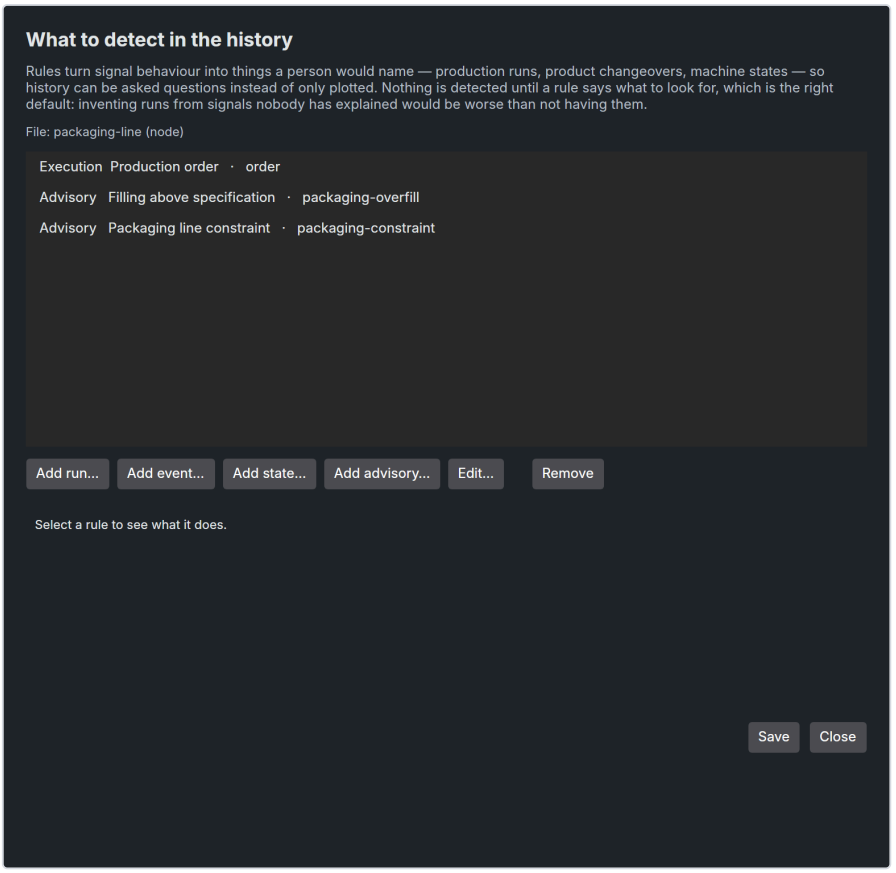
Where	Field	What to enter
Acquisition group	**Kind of asset**	What kind of thing this group is. The box offers whatever the catalogue declares. Leave it blank while nobody has decided.
Tag	**Role**	What this tag is to its asset, in the project's own words (the hint shows `e.g. transformer.oilTemperature`). It sits beside the tag's **Unit**.

An asset declared in the asset model with a **Kind** counts as well; the group's box is simply the quickest way to classify a machine that is one acquisition group.

Detection rules: runs, events, states and advisories

Detection rules teach the node to recognise things in what it acquires. Nothing is detected until a rule says what to look for, which is the right default: runs invented from signals nobody has explained would be worse than none. Detection runs on the node, queues its writes, and never delays acquisition.

Acquisition ▸ Detection Rules… needs the permission to configure the historian. It opens the rules of the node this Desktop runs or is joined to: they are part of that node's configuration, so they are read and saved through it as you, the person signed in, and the node checks your permission and records the change under your name.



The detection rules window, titled What to detect in the history, listing an execution and two advisories with the buttons to add, edit and remove rules (captured on an earlier version of the demonstration line)

Element	What it does
File:	Where these rules are kept, for example `packaging-line (node)`.
Rule list	One line per rule: its kind (**Execution**, **Event**, **State**, **Advisory**), its name and its id.
Add run…	Adds an execution rule.
Add event…	Adds an event rule.
Add state…	Adds a state rule.
Add advisory…	Adds an advisory rule.
Edit…	Edits the selected advisory. Runs, events and states are small enough to remove and add again, so only an advisory can be edited here.
Remove	Removes the selected rule.
Detail pane	What the selected rule does, in words. "*Select a rule to see what it does.*" until one is selected.
Save	Writes the rules to the node. They become part of the configuration revision, so they draft, diff, activate, roll back and audit like everything else.
Close	Closes the window. Unsaved changes are discarded.

If the rules file cannot be read, the window refuses to edit rather than start from an empty list, because saving would replace rules that are still there.

Run, event and state rules

Add run…, **Add event…** and **Add state…** open one dialog, titled *New rule — <kind>*, whose sections change with the kind.

Section / field	Kinds	Meaning
Id	all	The rule's id, for example `order`. Datasets and reports name the type by it.
Name	all	The readable name, for example *Production order*.
Asset	all	The asset the entity belongs to, for example `Packaging/Line`.
Starts when *(run, event)* / **The signal whose value is the state** *(state)*	all	The signal to watch, chosen from the project's signals (never typed, so a rule cannot watch a topic nobody publishes).
Comparison and **Value**	run, event	The start condition: `== true`, `== false`, `>`, `>=`, `<`, `<=`, `==` or `!=`, and the value to compare with where one is needed.
Hold for	run, event	Seconds or `hh:mm:ss`. The condition must hold this long before it counts. A signal crossing a threshold for one poll is noise; without this a chattering signal shreds a shift into hundreds of intervals.
Ends when: **Signal**, **Comparison**, **Value**	run	The end condition.
Give up after	run	A run that never ends is closed as *timed out* after this long rather than left open for ever. Blank means no limit.
What kind of event: **Severity**	event	`Unspecified`, `Information`, `Warning`, `Error` or `Critical`.
It has a duration	event	Off for a moment (a case rejected at the checkweigher); on for something you would ask how long it lasted (a fault).
What to call each value	state	One line per value: `value = name`, for example `true = Producing` and `false = Stopped`. Empty uses the raw values.
Ignore changes shorter than	state	A change shorter than this is not a state change but a chattering signal.
What describes it	run, event	Attributes recorded with each entity, one per line: `Name = signal, sampling`, where sampling is `AtStart`, `AtEnd`, `Last`, `Minimum`, `Maximum` or `Average` (default `AtStart`). A quoted value is a constant, for example `line = "Packaging 1"`. A set-point is what it was at the start; a quality figure is its average; a limit is the extreme.

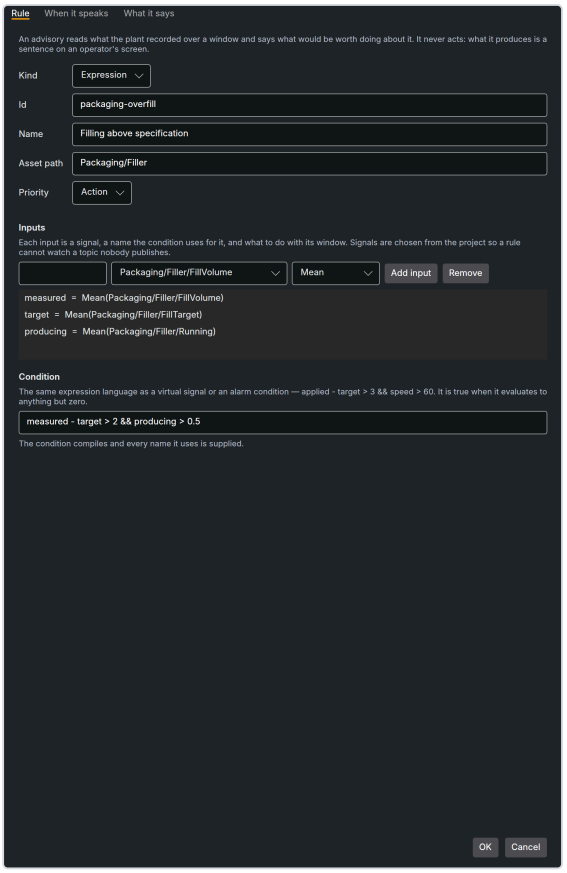
The packaging demonstration declares these rules:

Kind	Id	Name	Asset	Rule
Execution	`order`	Production order	`Packaging/Line`	Starts when `Line/ChangeoverActive` becomes false, ends when it becomes true, 3 s hold, gives up after 1 h 30 min; records `sku`, `ordered` at start, `good`, `rejects` at end, `meanRate` and `peakRate` of `Loader/Rate`
Execution	`changeover`	Changeover	`Packaging/Line`	The opposite edges of the same signal

Kind	Id	Name	Asset	Rule
Execution	`filler-run`	Pouch loader run	`Packaging/Loader`	While `Loader/Running` is true; records the mean, minimum and maximum `Checkweigher/CaseWeight`
Event	`filler-fault`	Pouch loader fault	`Packaging/Loader`	`Loader/Faulted` becomes true, 2 s hold, Warning, with a duration
Event	`capper-fault`	Checkweigher fault	`Packaging/Checkweigher`	`Checkweigher/Faulted` becomes true
Event	`depalletiser-starved`	Case erector waiting for blanks	`Packaging/Erector`	`Erector/Starved` becomes true, Information
Event	`palletiser-blocked`	Palletiser blocked at discharge	`Packaging/Palletiser`	`Palletiser/Blocked` becomes true, Information
State	`line-mode`	Line mode	`Packaging/Line`	`Line/ChangeoverActive`: true = Changeover, false = Producing; ignore changes under 5 s
State	`filler-state`	Pouch loader	`Packaging/Loader`	`Loader/Running`: Producing or Stopped
State	`capper-state`	Checkweigher	`Packaging/Checkweigher`	`Checkweigher/Running`: Producing or Stopped
Advisory	`packaging-constraint`	Packaging line constraint	`Packaging/Line`	Constraint over six machines
Advisory	`packaging-overweight`	Cases above weight specification	`Packaging/Checkweigher`	Expression, see below

Advisory rules

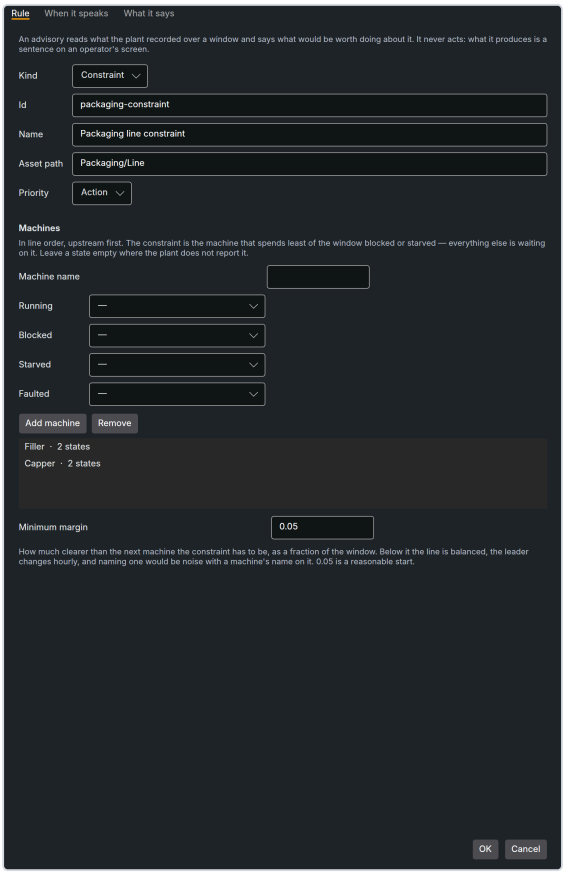
An advisory is the fourth kind of rule and the only one that produces a *recommendation* rather than a record. Runs, events and states answer what happened; an advisory answers what would be worth doing about it, judged over a window rather than at an instant: a line whose constraint has been the same machine all hour, cases running heavy on average while never tripping a limit. It never acts; what it produces is a sentence on an operator's screen.



An Expression advisory in the Advisory rule dialog: three inputs averaged over the window, the condition checked as it is typed, and the settings that decide when it speaks (captured on an earlier version of the demonstration line, with a filler instead of a checkweigher)

Section / field	Meaning
Kind	**Expression** (named inputs and a condition over them; the general case) or **Constraint** (the machines of a line, upstream first).
Id, **Name**	The rule's id and readable name.
Asset path	The asset the advisory is about. It appears on the pages scoped to that asset.
Priority	**Information** or **Action**.
Inputs *(Expression)*	Each input is a name the condition uses, a signal chosen from the project, and what to do with its window: `Mean`, `Minimum`, `Maximum`, `Last`, `Delta` (last minus first), `TimeTrue` (the fraction of the window a signal was non-zero, for booleans) or `StandardDeviation`. **Add input** and **Remove** edit the list.
Condition *(Expression)*	In the same expression language as a virtual signal or an alarm, for example `measured - target > 8 && producing > 0.5`. True when it evaluates to anything but zero. It is **compiled as you type**: "*The condition compiles and every name it uses is supplied*", or "*Nothing supplies X, so this condition could never be true*".
Machines *(Constraint)*	In line order, upstream first: **Machine name** and its **Running**, **Blocked**, **Starved** and **Faulted** signals (leave a state empty where the plant does not report it). **Add machine** / **Remove**. The constraint is the machine that spends least of the window blocked or starved.
Minimum margin *(Constraint)*	How much clearer than the next machine the constraint has to be, as a fraction of the window. `0.05` is a reasonable start.
Window	How far back the rule looks, for example `01:00:00`.

Section / field	Meaning
Evaluate every	How often it is evaluated.
Agreeing evaluations	How many evaluations in a row must agree before it says anything.
Re-arm after clearing	How long it stays quiet after clearing.
Believe a state for at most	How long a state signal's last value may be held before its time stops counting, so a recorder outage does not become an hour of confident *running*.
Subject	What the sentence is about, for example `Checkweigher`.
String key	An entry in the HMI project's string table, so the sentence is read in the operator's own language; `{0}` in it is replaced by the subject. Empty: the product writes the sentence in English.
Suggested action	Optional, and most rules should leave it empty: **Connection**, **Tag**, **Value to write** and **Button label**, plus **Ask the operator to confirm before sending**. It becomes a labelled button beside the sentence, pressed by a person and sent through the same permission, confirmation and audit as every other command. The HMI's advisory panel always asks the operator to confirm, even with that box cleared, because its button appears on its own wherever the operator is working, so the box has no effect there. Fill in all four boxes or none.



A Constraint advisory: the machines of the line in order, the minimum margin, and the same timing settings (captured on an earlier version of the demonstration line)

The four timing numbers decide whether an advisory is worth reading. One that appeared and vanished every minute would teach a shift to stop reading the panel it appears in. On the packaging line, `packaging-overweight` averages `CaseWeight`, `WeightTarget` and the fraction of time `Loader/Running` was

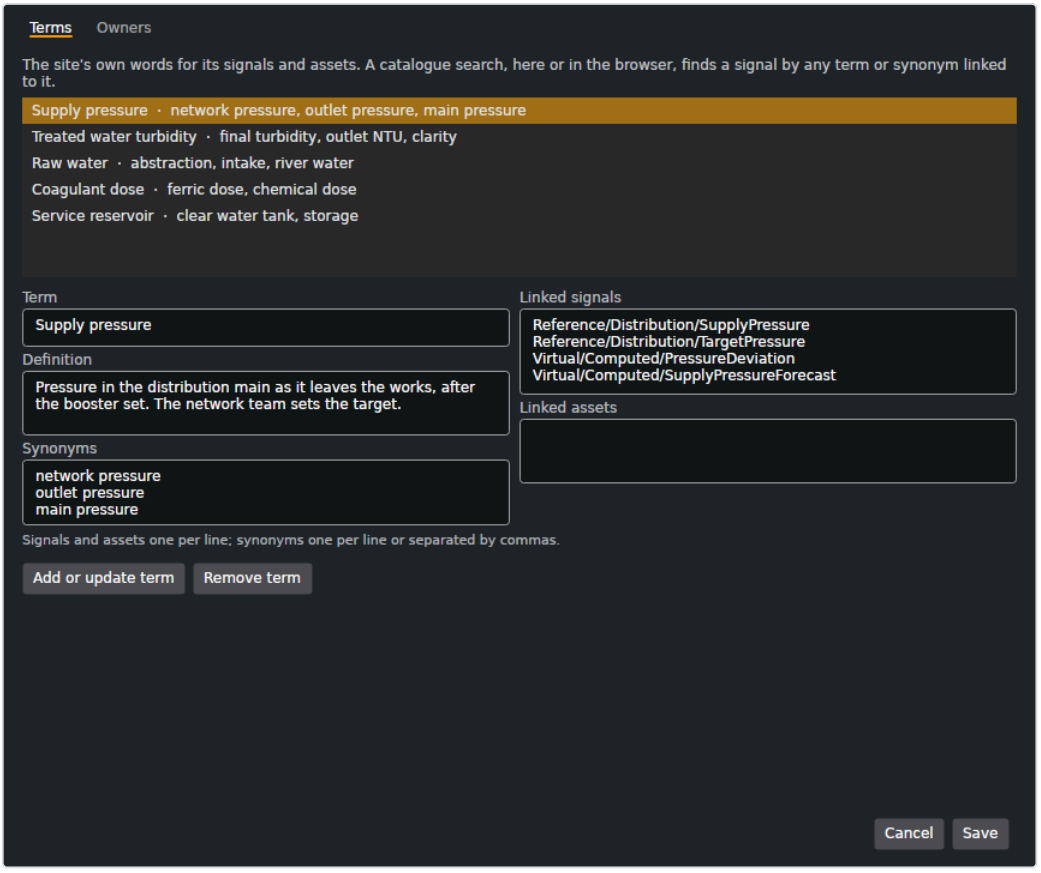
true over one hour, evaluates every five minutes, needs two agreeing evaluations and re-arms after thirty minutes; its condition is `measured - target > 8 && producing > 0.5`, cases 8 g heavier than target on average while the line was producing for more than half the hour.

The site's own words: glossary, owners and saved searches

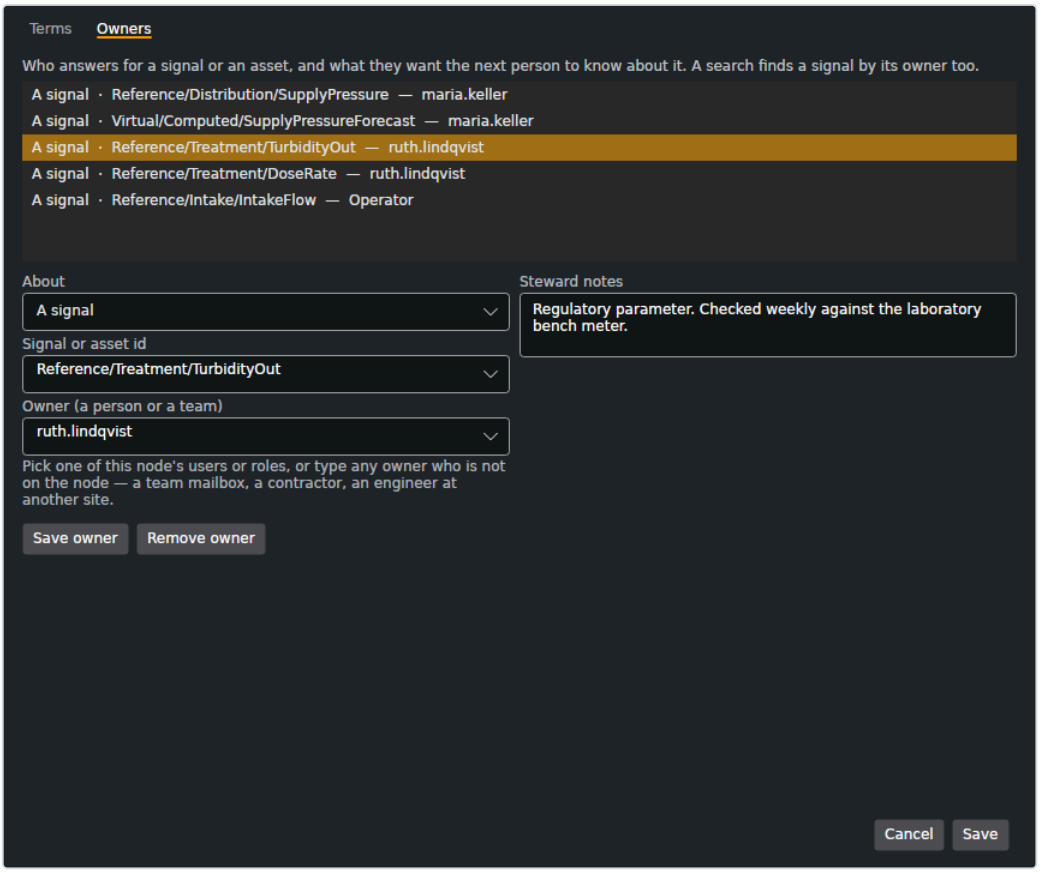
A plant's tag names are the integrator's; the words its people use are the plant's. The **glossary** links the two, and says who answers for each signal and asset.

Data ▸ Glossary and owners… opens a window with two tabs:

- **Terms** — the site's terms. Each has a **Term**, a **Definition**, **Synonyms** (one per line or separated by commas: *TS*, *strip surface temp*), and the **Linked signals** and **Linked assets** it is about, one id per line. Select a term in the list to load it into the form; **Add or update term** writes the form back (a term of the same name, ignoring case, is replaced); **Remove term** takes the selected one out.
- **Owners** — for a signal or an asset, its **Owner** (a person or a team, in your own words — often somebody who never signs in, such as *Line 1 maintenance*) and **Steward notes** (*Transmitter replaced 2026-09; reads 0.3 °C high until recalibrated*). **About** chooses a signal or an asset, and the id box offers the project's own. When you may manage users, the **Owner** box offers this node's users and the roles they hold, so a person on the node is spelt the way the node spells them; you can still type anybody who is not on the node — a team mailbox, an engineer at another site — and the window says so beneath the box. Without that permission nothing is offered and the window asks you to type the owner. A blank owner and blank notes take the entry out.



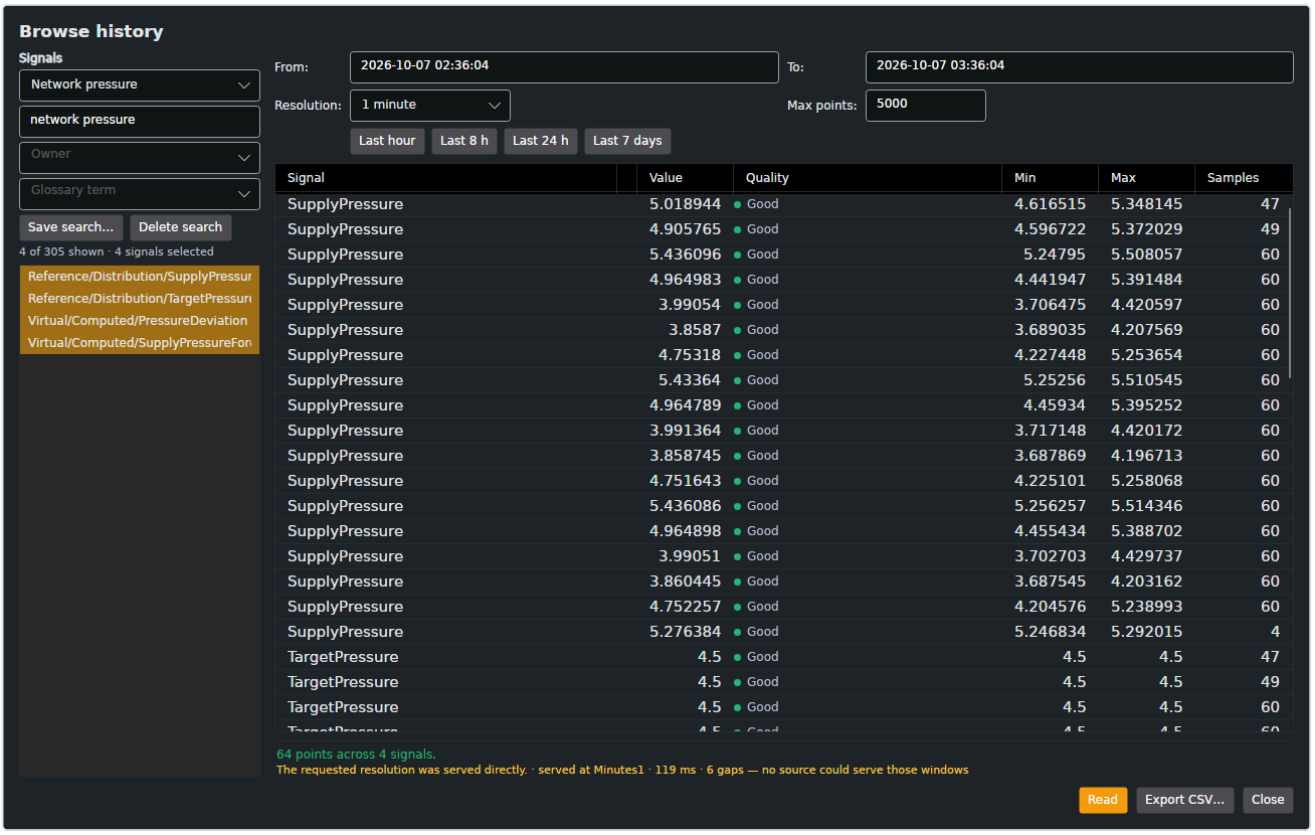
Data ▸ Glossary and owners, Terms tab, on the reference works: five terms with their synonyms, and Supply pressure loaded into the form with its definition, three synonyms and the four signals it links



The Owners tab: who answers for five of the works' signals — two process engineers on the node and the Operator role — with their steward notes

Save keeps it in the project, beside the asset model and the role catalogue: it is applied, activated, rolled back and audited with the rest of the semantic section, and the node's catalogue searches read it from the activated revision. A glossary this build cannot read is reported when the window opens and is never saved over. It is part of every edition.

Searched, not only shown. Data ▸ Browse History… filters its signal list with the same rule as every catalogue search in the product — the browser's **Catalogue** page, the data API, Grafana and an assistant over MCP: a signal is found by its id, name, description or unit, **and** by a linked term, a synonym or its owner. Beside the filter are an **Owner** and a **Glossary term** box, each matched whole. **Save search…** keeps the three under a name in **Saved searches**, for you: in this Windows account's own settings, under the name you are signed in to the node with, so two engineers who share a control-room login keep their own. A saved search is a shortcut, not configuration, and is in no revision.



Data ▸ Browse History with the saved search Network pressure chosen: the filter "network pressure" matches no tag name and finds, through the glossary's synonym, the four signals linked to Supply pressure, read in one-minute buckets

Moving a semantic model between sites

Data ▸ Export Semantic Model… writes the site's semantic model to one JSON file (default name `semantic-model.json`): the **detection rules**, the **role catalogue** and the **glossary** together, because half a model is not a model. A file with a glossary in it is format 2, which a build from before the glossary refuses by name rather than importing the rules and dropping the site's vocabulary; a file without one is format 1, which every build reads. The asset model travels with the project instead. Like the rules editor, it works on the node this Desktop runs or is joined to — on a workstation joined to a node, the rules come from and go to that node's configuration.

If the installation detects nothing and declares no kinds of asset, the export says there is nothing to export yet. After a successful export the event log reads, for example, "*Semantic model exported to D:\semantic-model.json. Detection rules: 12. Kinds of asset: 1.*"

Data ▸ Import Semantic Model… reads such a file back:

1. Choose the file. It is parsed and validated first; a file this build cannot use is refused with the reason, and nothing is changed.
2. Confirm. The message says what the file holds and where it will go: "*This file holds — detection rules: 12, kinds of asset: 1. Importing replaces the rules at … and this project's role catalogue: both together, or neither.*" Press **Import**.
3. The rules are written first, because they are the half a node can refuse; if the node refuses them, the role catalogue is left as it was. Then the role catalogue replaces the project's, and so does the glossary when the file carries one (a file without a glossary leaves the project's as it is).

Save the project afterwards so the imported catalogue is kept. Both export and import need the permission to configure the historian.

Runs from a production database: run sources

Detection rules find runs in signals. Many plants already know their runs without that: the MES or level-2 system writes a row per batch, order or lot into a production database, with its SKU, its counts and its weights. A **run source** reads that database on the node and writes each row into the semantic model as an execution:

Row	Run
the key column	the run's id, as `{source}:{key}`, so a row read again updates its run
the identifier column	the run's identifier (the batch number); the key when there is none
the start and end columns (or start and duration)	the run's start and end, in UTC; no end is a run still going
the asset column, or a fixed asset	the run's asset
a fixed run type, or a type column	the run's execution type
each attribute column	an attribute: text, or a number kept as a number

Runs from a source are ordinary executions: they appear wherever detected runs do — run comparison, datasets with one row per run, run reports, the API — and their numeric attributes are numbers an analysis can compute with. A source reads only with a single SELECT, a bounded number of rows at a time, and each read continues from where the last stopped. Run sources are set up in the browser under **Administration ▸ Run sources**; the *Administration in the browser* chapter describes the screen, its test before saving, and a packaging-hall example.

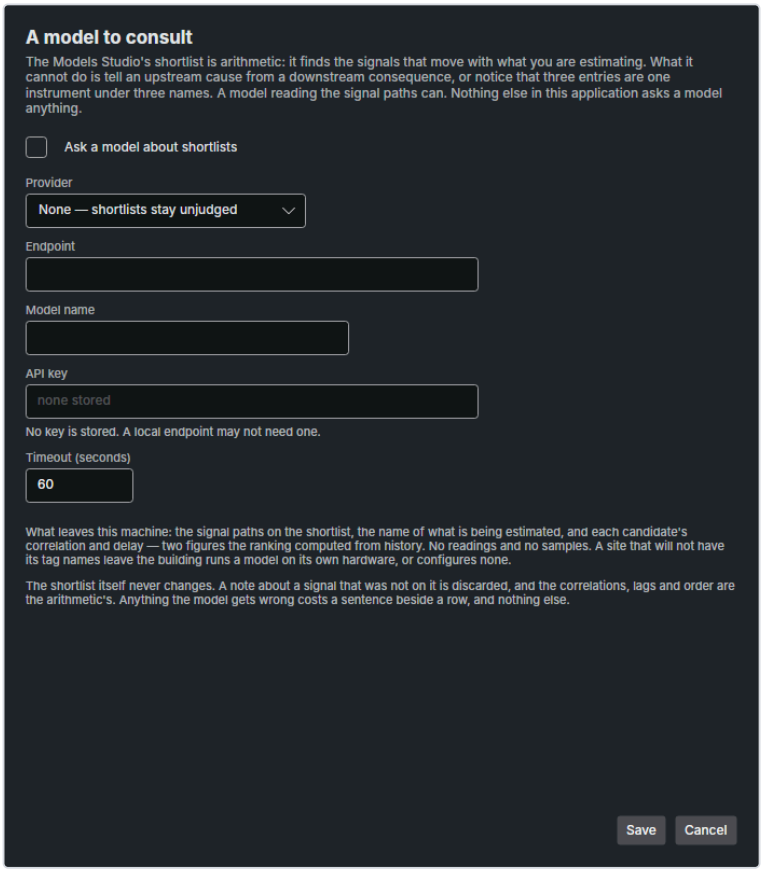
How the semantic model is used

Once rules are running, what they detect is used throughout the product:

- **Trends.** Right-click a Desktop trend ▸ **Process context** and choose a state type (for example *Line mode*): the chart is shaded with the intervals the plant spent in each state. **None** removes the shading.
- **Datasets.** A dataset can be segmented by an execution, event or state type (one segment per order, a window around each fault, one segment per *Producing* interval), and its shapes include event tables, state intervals and one row per run. A column can name a **role** instead of a signal and resolve against whichever asset the dataset is pointed at. See the *Datasets* chapter.
- **Reports.** A report's scope can be a calendar period or one section per run: the demonstration's run report lists every production order of the last 24 hours.
- **Analysis.** Run comparison aligns executions on their phase rather than the clock, and the Analysis Studio uses the same entities to find what changed.
- **Operator screens.** With the Web HMI module, the timeline and production components and the advisory panel on operator pages read the same model.
- **The API and the assistant.** The data-access API serves assets, types, executions, events, states and context (a semantic query is bounded to 31 days and 500 rows, and one filtering on attributes must give a time window). With the Studio Assistant module, the assistant can list the plant's assets and semantic types when it helps an author.

Data ▸ Model for shortlist advice

The **Data** menu also holds **Model for shortlist advice…**, the setting for an optional outside model that comments on the Models Studio's shortlist of candidate signals. It matters only on installations with the Models module; with **None — shortlists stay unjudged** (the default) nothing leaves the machine.



The Model for shortlist advice window with no model configured: the Ask a model about shortlists switch, provider, endpoint, model name, API key and timeout

Field	Meaning
Ask a model about shortlists	Turns the consultation on.
Provider	**None — shortlists stay unjudged**, a hosted provider, or an OpenAI-compatible endpoint (for a model the site runs on its own hardware).
Endpoint	The address of a compatible endpoint.
Model name	Which model to ask.
API key	Stored sealed; *none stored* until one is entered. A local endpoint may not need one.
Timeout (seconds)	How long to wait, `60` by default.

What leaves the machine is the signal paths on the shortlist, the name of what is being estimated, and each candidate's correlation and delay; no readings and no samples. The shortlist itself never changes: anything the model gets wrong costs a sentence beside a row, and nothing else.

Worked example: the Checkweigher as a kind of asset

The packaging line has one checkweigher, acquired as the group `Checkweigher` under the connection `Packaging`, with tags including `CaseWeight` (g), `WeightTarget` (g), `Running` and `Faulted`. Out of the box it is registered as the asset `Packaging/Checkweigher · Area`, which says nothing about what it is. The steps below describe it properly, so that a dataset written for *a checkweigher* works on this one and on the next line's.

1. Declare the kind and its roles.

1. Choose **Data ▸ Role catalogue…**.
2. In **Name** (left) type `Checkweigher`; in **Description** type `In-line checkweigher that weighs every case`. Press **Add kind**.
3. With *Checkweigher* selected, add three roles. For each, fill in the fields on the right and press **Add role**:

Role	Unit	Required	Description
`checkweigher.caseWeight`	`g`	ticked	Weight of each case as it crosses the scale
`checkweigher.weightTarget`	`g`	ticked	Nominal case weight for the running SKU
`checkweigher.running`		cleared	True while the checkweigher is producing

4. Press **Check this plant**. Nothing is reported yet: no asset has the kind *Checkweigher*, so none is examined.
5. Press **Save**.

2. Classify the asset.

1. Choose **Data ▸ Asset model…**.
2. In **What the node would register**, select `Packaging/Checkweigher · Area` and press **Adopt**. It appears under **Declared here** with **Id** `Packaging/Checkweigher`.
3. Set **Name** to `Checkweigher`, **Under** to `Packaging`, **Kind** to `Checkweigher` (offered from the catalogue), **Description** to `Line 1 checkweigher`, and keep **In service** ticked. Press **Apply**.
4. The right-hand list now shows `Packaging/Checkweigher · Checkweigher`. Press **Save**.

(Setting **Kind of asset** to `Checkweigher` on the `Checkweigher` group in the I/O configuration has the same effect for the check.)

3. Bind the roles.

1. Choose **Acquisition ▸ I/O Configuration…** and select the tag `CaseWeight` in the `Checkweigher` group.
2. In **Role**, beside **Unit** `g`, type `checkweigher.caseWeight`.
3. Select `WeightTarget` and set its **Role** to `checkweigher.weightTarget`.
4. Apply the configuration.

4. Check the binding. Open **Data ▸ Role catalogue…** again and press **Check this plant**. `checkweigher.running` is not bound, but it is optional, so the list shows it without the required mark; the two required roles are satisfied. Bind `Running` too (**Role** `checkweigher.running`) and the check reads "*Every declared role is bound on every instance.*"

5. Save and activate. Save the project (**File ▸ Save**) and activate it so the node registers the new asset and roles.

The result: a dataset column that names the role `checkweigher.caseWeight` instead of the signal `Packaging/Checkweigher/CaseWeight` produces a column headed `checkweigher.caseWeight` on every checkweigher it is pointed at, and the detection rule *Checkweigher fault* and the advisory *Cases above weight specification*, both on the asset `Packaging/Checkweigher`, now belong to an asset that says what it is.

Datasets

A **dataset** is a saved, reusable definition of an extract from history: which signals become columns, how rows are placed in time, what happens where a value is missing, and how many rows it may produce. It carries no time range; you give that each time you run it. Define it once, and every report, script or analyst that needs *those* numbers asks for the dataset by name instead of rebuilding the query.

This chapter explains what a dataset definition contains, how to build one in **Data ▸ Dataset Studio…**, how datasets are run and exported, the nine worked examples the packaging demonstration installs, and a step-by-step example.

Datasets and reports are part of the base product.

Why datasets exist

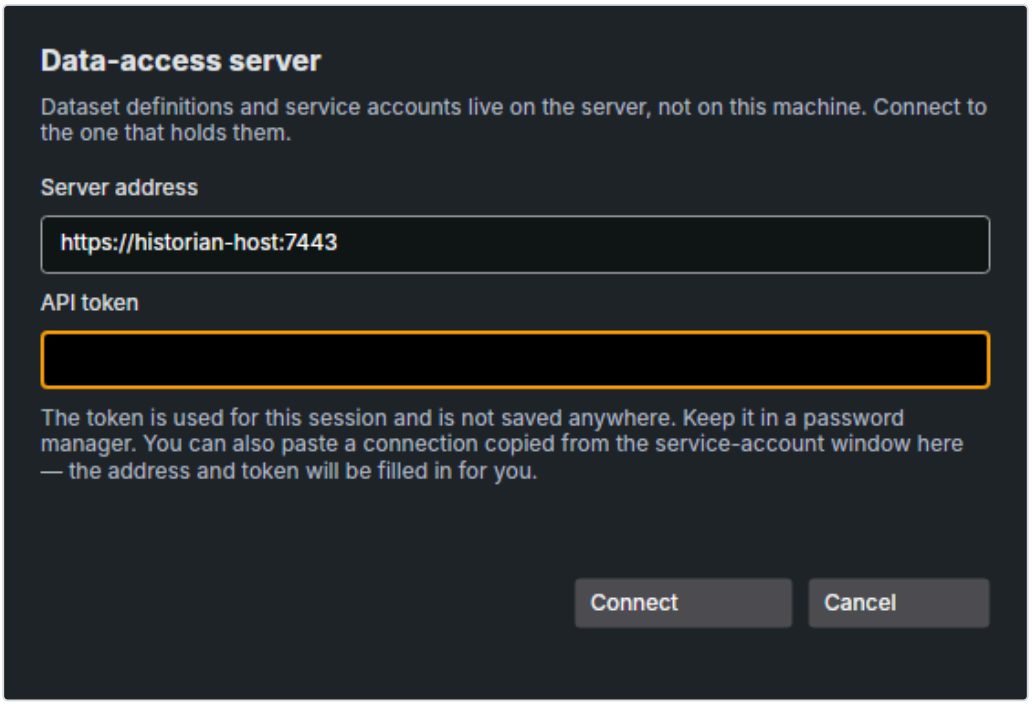
A trend answers *what did it look like*. A dataset answers *give me the numbers*, in a shape a table, a spreadsheet, a report or a dataframe can use. Three properties make it the right way to hand data to somebody:

- **By name.** Report elements and integrations bind to a dataset **by name**, never to a historian, a table or a credential. The same definition works on any installation that has it, whichever storage provider holds the history.
- **Server-side.** The definition is kept by the node, with its configuration. It survives the person who wrote it and is the same for everybody who runs it.
- **Honest about gaps.** Where a value is missing or a result was cut short, the dataset says so (an empty cell, a truncation flag, warnings) rather than inventing a number.

Where datasets live, and how the Desktop reaches them

Dataset definitions belong to the **node**, not to the project file, so they are the one part of a plant that does not travel in a `.meelmoproj`. The Desktop reads and writes them through the node's **data-access API**:

- On a workstation that **hosts its own node** (a standalone installation), the node serves its API to this Desktop on the local loopback address only, with a credential minted for the person signed in. Nothing is asked; the Dataset Studio simply opens.
- On a workstation that is a **client of a node elsewhere**, the Desktop uses the credential it was given when it was attached to that node, and the address the node publishes. Again nothing is asked.
- Only when neither is available does the Desktop ask for a data-access server: the **Data-access server** dialog below. The connection is kept for the rest of the session and used by the Dataset Studio, the Report Studio, the Analysis and Models Studios and **Administration ▸ Service Accounts…**.



The Data-access server dialog, asking for the server address and an API token

Field	Meaning
Server address	The data-access server that holds the definitions, for example `https://historian-host:7443`.
API token	A service-account token with the permissions you need. It is used for this session and is not saved anywhere. You can paste a connection copied from the service-account window here; the address and token are filled in for you.
Connect / **Cancel**	Connects, or gives up. "*The server could not be reached.*" means the address, the network or the certificate is wrong.

A data-access server reachable over the network by other machines and scripts, and the Python and .NET SDKs, are the *Data access and SDKs* module; the loopback connection a standalone Desktop uses to reach its own node is part of the base product.

What the server allows depends on the account: listing definitions needs `ReadCatalog`, saving one needs `ManageDatasets`, running one needs `ReadHistory`, and exporting to a file needs `Export`.

Saving from the Dataset Studio and the Report Studio is done as you. On a workstation that hosts its node, or one attached to a node elsewhere, a dataset or a report saved or removed in the two Studios is saved at the node as the person signed in to this Desktop — not with the workstation's own credential, which on an attached workstation may only read. The node checks your permissions to change and activate its configuration (`Config.Io` and `Config.Activate`; an engineer has both) and records the revision under your name. With nobody signed in, or with a server typed into the dialog above, the Studios save with the token as before, and the server checks the token's account — `ManageDatasets` — and records the save under that account.

What a dataset definition contains

Shape: how rows are laid out

Shape	One row per	Columns	Use it for
`Wide`	timestamp	a timestamp, then one column per signal	Tables, trends, spreadsheets: what most people mean by *the data*
`Long`	sample: (timestamp, signal, value)	timestamp, `signal`, value, quality	Loading a database or a dataframe
`EventTable`	event the detection rules raised	the event's time and what the rule recorded about it	*What was raised* sections; needs no signal columns
`ExecutionSummary`	run (execution)	per signal: the run aggregates chosen (average, minimum, maximum, count by default)	One row per order, batch or pallet
`StateIntervals`	state interval	the state, when it started and ended	*Where the time went*; needs no signal columns
`Sequence`	position within a segment	rows numbered from the start of each run	Overlaying runs on elapsed time instead of the clock

`EventTable`, `ExecutionSummary` and `StateIntervals` read the semantic model: they return rows only where the detection rules declare and have detected events, runs or states. See the *semantic model* chapter.

Sampling: how rows are placed in time

Sampling	Rows	Notes
`Original`	Every stored sample on its own timestamp; nothing invented	The row count is the sample count, bounded by **Maximum rows**
`FixedInterval`	One row per **Interval**	Resamples signals acquired at different rates onto one grid; needs an interval
`AcquisitionBatch`	One row per correlated acquisition batch: only instants the plant genuinely read these signals together	Only meaningful where the project declares a temporal coherence frame; the right choice for correlation

Alignment and the maximum gap

On a fixed interval, a row is often at an instant where a signal had no sample. **Alignment** decides what the row gets: `None` (nothing), `Previous` (hold the last value forward, the default), `Next`, `Nearest` or `Linear` (interpolate).

Alignment is a decision with consequences. `Previous` is right for a setpoint and wrong for a flow meter that stopped reporting. Bound it with a **maximum gap**: past that, the cell is missing rather than carrying a value for longer than it can be believed. A value interpolated between two samples carries the worse of their qualities. Whatever the maximum gap, no value is carried, looked ahead to or interpolated across a gap the historian reported — time the node was not acquiring the signal, or no store could serve: those cells are missing.

Missing values

Policy	What is written where no value is available
`Null`	An empty cell. The honest default: absent is not zero.
`DropRow`	The whole row is left out.
`Constant`	A constant named in the column's settings.

Columns

Each column reads one signal (for example `Packaging/Checkweigher/CaseWeight`) and has an output name, which defaults to the signal id. A column may instead name a **role** (for example `checkweigher.caseWeight`) that is resolved against the asset a run is about, so one definition serves every machine of a kind; it is then headed by the role. A column can also carry its own alignment, missing-value policy and a companion quality column. Two columns may read the same signal, but no two may share a name.

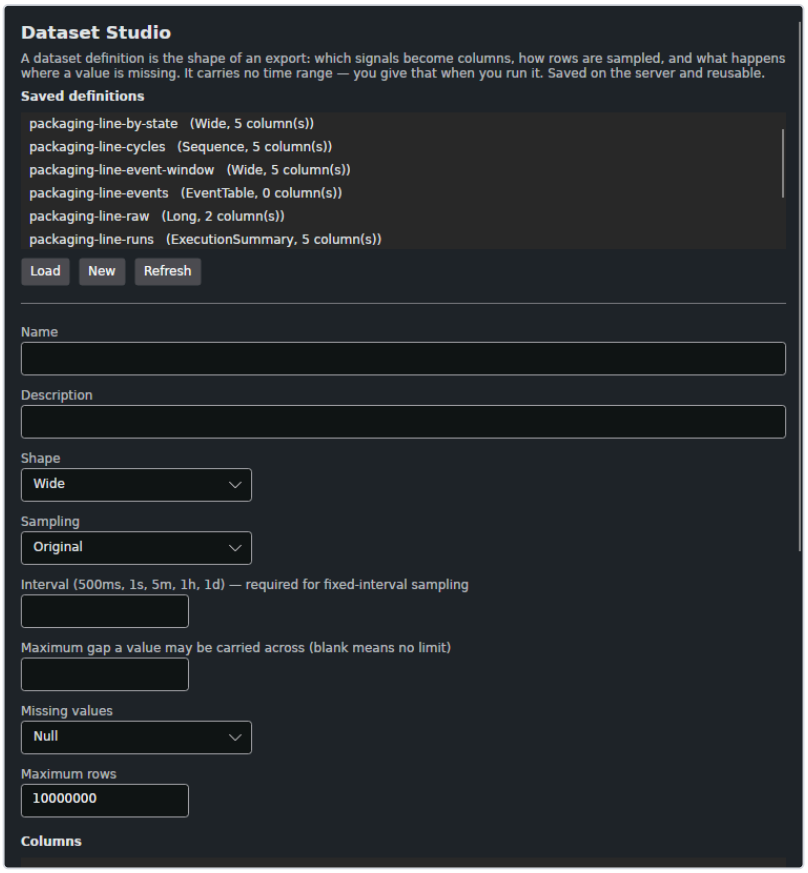
Settings beyond the Desktop form

The definition format is larger than the Dataset Studio's form. These settings are part of every definition, are kept when you load and re-save a definition in the Studio, and are set through the data-access API as JSON, or by starting from one of the demonstration definitions that already carries them:

Setting	Meaning
Alignment	Dataset default, overridable per column (see above).
Quality	`All`, `GoodOnly` or `ExcludeBad`: which samples are admitted.
Segmentation	`Calendar` (a plain range), `EventRelative` (one segment per event of a type, with optional lead-in and lead-out), `ExecutionRelative` (one per run of a type) or `StateInterval` (one per interval of a state type), optionally limited to an asset subtree, to segments carrying given context values, and to a minimum duration; at most 10,000 segments.
Transforms	Derived columns: resample, aggregate, difference, ratio, derivative, integral, lag, shift, rolling minimum/maximum/average/standard deviation, time above or below a threshold, time since an event or state change, elapsed and normalized progress of a run, value at run start or end, normalize, unit conversion.
Run aggregates	For `ExecutionSummary`: which reductions per signal (average, minimum, maximum, count, sum, first, last, RMS, standard deviation, delta, integral, slope, intercept, ...). `Delta` (last minus first) is the one a totalizer needs.
Context fields	Context values joined onto every row.
Sources	Which stores to read; empty reads the historian and everything read alongside it.

Data ▸ Dataset Studio

Data ▸ Dataset Studio... lists the definitions the node holds and edits one at a time. Its first line says what a definition is: "*A dataset definition is the shape of an export: which signals become columns, how rows are sampled, and what happens where a value is missing. It carries no time range — you give that when you run it. Saved on the server and reusable.*"



The Dataset Studio on the packaging node: the nine demonstration datasets under Saved definitions, with packaging-line-synchronised loaded so its shape, sampling and five columns show in the form

Saved definitions

Element	What it does
Saved definitions	Every definition the node holds, as `name (Shape, N column(s))`.
Load	Loads the selected definition into the form. Loading and saving unchanged gives back exactly the same definition, including the settings the form does not show.
New	Clears the form for a new definition. Nothing is saved until you save it.
Refresh	Reads the list again from the node.

The form

Field	Meaning
Name	The name reports and scripts use, for example `packaging-line-shift`. Required. Saving under an existing name replaces that definition.
Description	What the dataset is for, for whoever finds it next.
Shape	`Wide`, `Long`, `EventTable`, `ExecutionSummary`, `StateIntervals` or `Sequence`.
Sampling	`Original`, `FixedInterval` or `AcquisitionBatch`.
Interval (500ms, 1s, 5m, 1h, 1d) — required for fixed-interval sampling	The row spacing, written as `500ms`, `1s`, `5m`, `1h` or `1d`.

Field	Meaning
Maximum gap a value may be carried across (blank means no limit)	Past this gap a cell is missing rather than filled by alignment, for example `5m`.
Missing values	`Null`, `DropRow` or `Constant`.
Maximum rows	The upper bound on rows, `10000000` by default. It applies to running and exporting.
Columns	The columns in order, as `1. name ← signal`. In a wide dataset this order is the column order of every file it produces.
Move up / **Move down** / **Remove**	Reorder or remove the selected column.
Signal	The signal a new column reads. It proposes the node's stored signals and this project's configured signals as you type, so a column is chosen rather than remembered; the line beneath it says where the proposals came from.
Column name (blank uses the signal's own)	The header of the new column.
Add column	Adds the column.
Save definition	Validates and saves to the node. Every problem is listed at once, for example "*Fixed-interval sampling requires a positive interval.*", "*A wide, long or sequence dataset needs at least one column.*" or "*Duplicate column name 'Rate'.*" A successful save shows *Saved.* and refreshes the list.

The column list is meaningless for `EventTable` and `StateIntervals`, whose rows are occurrences and intervals rather than signals: leave it empty for those shapes.

Running a dataset over a time range

A definition has no time range; the range is given when it is run. In the base product a dataset is run by:

- **A report.** Every report section (table, chart, events, states, summary) names a dataset and runs it over the report's period: the last hour, the last 24 hours, one run. **Data ▸ Report Studio…** previews a report with its datasets run against the historian, and a scheduled report runs them on the node and writes the result as HTML, Excel (`xlsx`) or PDF. This is how a dataset becomes an Excel workbook. See the *Reports* chapter.
- **The data-access API.** `POST /api/v1/datasets/query` runs a definition, named or given in full, over a `start` and `end` and returns the rows, the schema, the row count, a `truncated` flag and warnings. `POST /api/v1/datasets/export` returns a file. Reaching the API from other machines and scripts is the *Data access and SDKs* module.

With the Web HMI module, the browser's report pages run the same datasets.

Export formats

Format	How	Notes
CSV	`format: "csv"` on the export call	The default
Apache Arrow	`format: "arrow"`	Licence-gated (Arrow access)
Apache Parquet	`format: "parquet"`	Licence-gated (Parquet export)

Format	How	Notes
Excel	A report with a table section over the dataset, rendered as `xlsx`	Reports also render HTML and PDF

All encodings of one definition have the same columns, so which format a pipeline chose never changes its schema. Dataset export itself must be included in the licence; an unlicensed export is refused with that reason. The server clamps **Maximum rows** to its own ceilings for queries and exports.

Limits and truncation

- A dataset whose rows are samples reads its window whole, up to ten million rows per signal. One that holds more is refused with the signal named, rather than exported from part of it.
- An event table, state table or execution summary counts **entities**, not samples, against its row limit, and reads the samples behind each row separately. A run too large to read is left **empty** rather than summarized from part of itself; the dataset is marked truncated and its warnings name what was left out.
- The value an event or state row carries is the one standing at its instant: on a fixed interval the row whose bucket holds it, otherwise one no older than the maximum gap. Past either, the cell is empty.

The demonstration datasets

Because definitions belong to the node, a fresh installation has none. **Help ▸ Demonstration…** installs worked examples: tick **Also install its saved queries and reports** when you open a demonstration plant. Definitions whose names already exist are left untouched, and the status line says how many were installed and how many were already there. On a client workstation the option is unavailable, because those definitions belong to the node's administrator.

The definitions are derived from the plant: their columns are the signals the plant's saved chart layout opens on, with up to two computed signals kept a place. A query that needs something the plant does not declare (a run, an event or a state type, or a temporal coherence frame) is not installed, because it would select nothing and look exactly like a plant that has not run. The packaging line (*Packaging line (simulated)*) declares all of them and receives all nine, named `packaging-line-…`:

Name	Shape	Sampling	Segmentation	Columns and settings	What it shows
`packaging-line-shift`	Wide	FixedInterval `1m`	Calendar	8 columns: `MagazineLevel`, `Erector Rate`, `BlanksPerMinute`, `BottomSealer Rate`, `SealPressure`, `SealTemperature`, `Availability`, `OEE`; alignment Previous, maximum gap 5 min, quality GoodOnly, missing Null; transform `MagazineLevel (15 min average)` (rolling average)	The table a shift report draws, and the rows a trend is drawn from

Name	Shape	Sampling	Segmentation	Columns and settings	What it shows
`packaging-line-raw`	Long	Original	Calendar	2 columns: `MagazineLevel`, `Erector Rate`; no alignment, all qualities; maximum 200,000 rows	Every sample exactly as recorded, for somebody else's tool
`packaging-line-events`	EventTable		Calendar	No columns; maximum 10,000 rows	A row per event the detection rules raised (loader and checkweigher faults, erector starved, palletiser blocked)
`packaging-line-states`	StateIntervals		Calendar	No columns; maximum 10,000 rows	A row per state interval and how long it lasted: where the time went
`packaging-line-runs`	ExecutionSummary		ExecutionRelative, type `order`	5 columns; aggregates average, minimum, maximum, standard deviation, delta, count; quality GoodOnly	One row per production order
`packaging-line-cycles`	Sequence	FixedInterval `10s`	ExecutionRelative, type `order`	5 columns; alignment Previous, maximum gap 1 min; maximum 100,000 rows	Every order as a curve numbered from its start, so orders overlay
`packaging-line-event-window`	Wide	FixedInterval `1s`	EventRelative, type `filler-fault`, lead-in 30 s, lead-out 2 min	5 columns; maximum gap 30 s; all qualities	What every signal did around each pouch loader fault
`packaging-line-by-state`	Wide	FixedInterval `30s`	StateInterval, type `line-mode`	5 columns; maximum gap 2 min; quality GoodOnly	The measurements cut by Producing and Changeover intervals
`packaging-line-synchronised`	Wide	AcquisitionBatch	Calendar	5 columns; no alignment, quality GoodOnly, missing DropRow; maximum 200,000 rows	Only instants the plant read these signals together, for correlation

The five-column definitions take the first five of the shift columns: `MagazineLevel`, `Erector Rate`, `BlanksPerMinute`, `BottomSealer Rate` and `SealPressure`. Where two chosen signals share a tag name (both `Rate`), the column name is qualified by group.

The two demonstration reports are built on them: `packaging-line-shift-report` (the last hour: a trend and a table of `packaging-line-shift`, the events and the states) and `packaging-line-run-report` (every order of the last 24 hours from `packaging-line-runs`). Read them, copy them, or delete them.

Worked example: a checkweigher dataset, step by step

The quality engineer wants one row a minute of the checkweigher's case weight against its target and the line's good count, for the last shift, as a spreadsheet.

Build the definition.

1. Choose **Data ▸ Dataset Studio…**. On a standalone installation it opens directly; otherwise connect to the node's data-access server when asked.
2. Press **New**.
3. **Name**: `checkweigher-1m`. **Description**: `Case weight against target, one row a minute, for the quality review`.
4. **Shape**: `Wide`. **Sampling**: `FixedInterval`. **Interval**: `1m`.
5. **Maximum gap a value may be carried across**: `2m`. A minute with no case (the line stopped) is filled from the previous minute at most twice, then left empty rather than repeating an old weight for an hour.
6. **Missing values**: `Null`. **Maximum rows**: leave `10000000`.
7. Add the columns. For each, type in **Signal** (pick the proposal), fill in **Column name**, and press **Add column**:

Signal	Column name
`Packaging/Checkweigher/CaseWeight`	`CaseWeight_g`
`Packaging/Checkweigher/WeightTarget`	`Target_g`
`Packaging/Line/GoodCount`	`GoodCases`

   The **Columns** list reads `1. CaseWeight_g ← Packaging/Checkweigher/CaseWeight`, `2. Target_g ← Packaging/Checkweigher/WeightTarget`, `3. GoodCases ← Packaging/Line/GoodCount`.
8. Press **Save definition**. The message reads *Saved.* and `checkweigher-1m (Wide, 3 column(s))` appears under **Saved definitions**.

If you had left **Interval** empty, saving would list "*Fixed-interval sampling requires a positive interval.*"; fix it and save again.

Run it over the last shift.

- In a report: in **Data ▸ Report Studio…** add a table section bound to `checkweigher-1m`, set the report period to the last 8 hours, preview it, and schedule or render it as `xlsx` for an Excel workbook of 480 rows.
- Through the API, with a token that may read history:

```
POST /api/v1/datasets/export
{ "definitionName": "checkweigher-1m",
  "start": "2026-09-29T06:00:00Z", "end": "2026-09-29T14:00:00Z",
  "format": "csv" }
```

The response is `checkweigher-1m.csv`: a timestamp column, then `CaseWeight_g`, `Target_g` and `GoodCases`, one row per minute.

Refine it. The engineer later wants the minutes cut by what the line was doing, so that an average over a Changeover interval is not mixed with one over Producing. That is a segmentation (`StateInterval`, type `line-mode`), which the Desktop form does not show: load `packaging-line-by-state`, which already carries it, remove its columns and add the three above, change **Name** to `checkweigher-by-mode-1m` and **Interval** to `1m`, and press **Save definition**. The new definition keeps the segmentation, quality filter and alignment it was loaded with, and produces one segment per Line mode interval, each Producing and each Changeover interval on its own.

Reports

A report is a document about the plant: a trend of the last shift, the readings behind it, what was raised and where the time went, laid out so that it can be printed, mailed and filed. This chapter covers how reports are defined in the Desktop's **Report Studio**, what each part of a definition means, how a report is previewed, scheduled and delivered, and the two reports the demonstration installs.

Reports are built on datasets. Every section of a report that shows data reads a saved dataset by name, so the *Datasets* chapter is the one to read first if you have not created one yet.

What a report is

A report definition answers four questions, and the Report Studio has one tab for each:

Question	Where it is answered	Example on the packaging line
What is it about?	**Definition** tab: the scope	the last 8 hours of the whole line, or every production order of the last day
What does it show?	**Sections** tab: the sections, in reading order	a note, a trend, a table of readings, the events raised, the line states
When does it run by itself?	**Schedule** tab: the schedule	at 06:00, 14:00 and 22:00, Europe/Berlin time
Where does it go?	**Schedule** tab: the delivery	written to disk and mailed to the shift leaders

Four properties explain most of how reports behave:

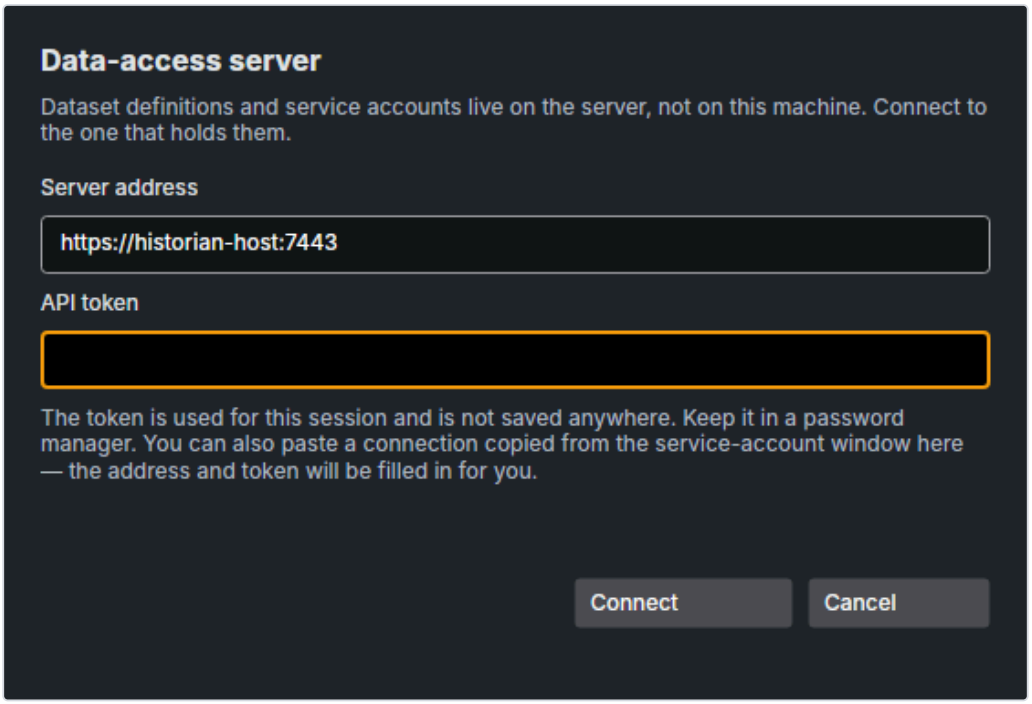
- **A report is saved on the node, not on your workstation.** Report definitions, like saved datasets, belong to the node's configuration. A report you save from one workstation is the same report everybody else sees, and it is the one the node runs on its schedule. It survives the person who wrote it.
- **A report binds datasets by name, never a database.** A definition names saved datasets such as `packaging-line-shift`. It carries no historian address, no password and no query text, so a definition is portable between installations that hold the same datasets.
- **Every report states its coverage.** Each report says what it could not see, for example "8 of 9 readings recorded; 1 not". This line cannot be switched off. A report that silently averaged over the readings it had would be a report somebody signs without knowing it was incomplete.
- **A missing value stays missing.** A field with no reading prints an em dash (`–`), never zero. A zero and a gap are different facts about the plant.

Opening the Report Studio

Data ▸ Report Studio… opens the studio.

The Report Studio is a client of the node's data-access service, because that is where report definitions are kept. How it connects depends on the installation:

- On a workstation that hosts its own node, it connects to that node directly and asks nothing.
- On a workstation attached to a node elsewhere, it uses the credential the workstation was given when it was attached, and asks nothing.
- Otherwise it asks once per session for the **Server address** and an **API token**. The token is used for the session and is not saved anywhere. You can also paste a connection copied from the service-account window and both fields are filled in.



The data-access connection dialog, asking for the node's server address and an API token

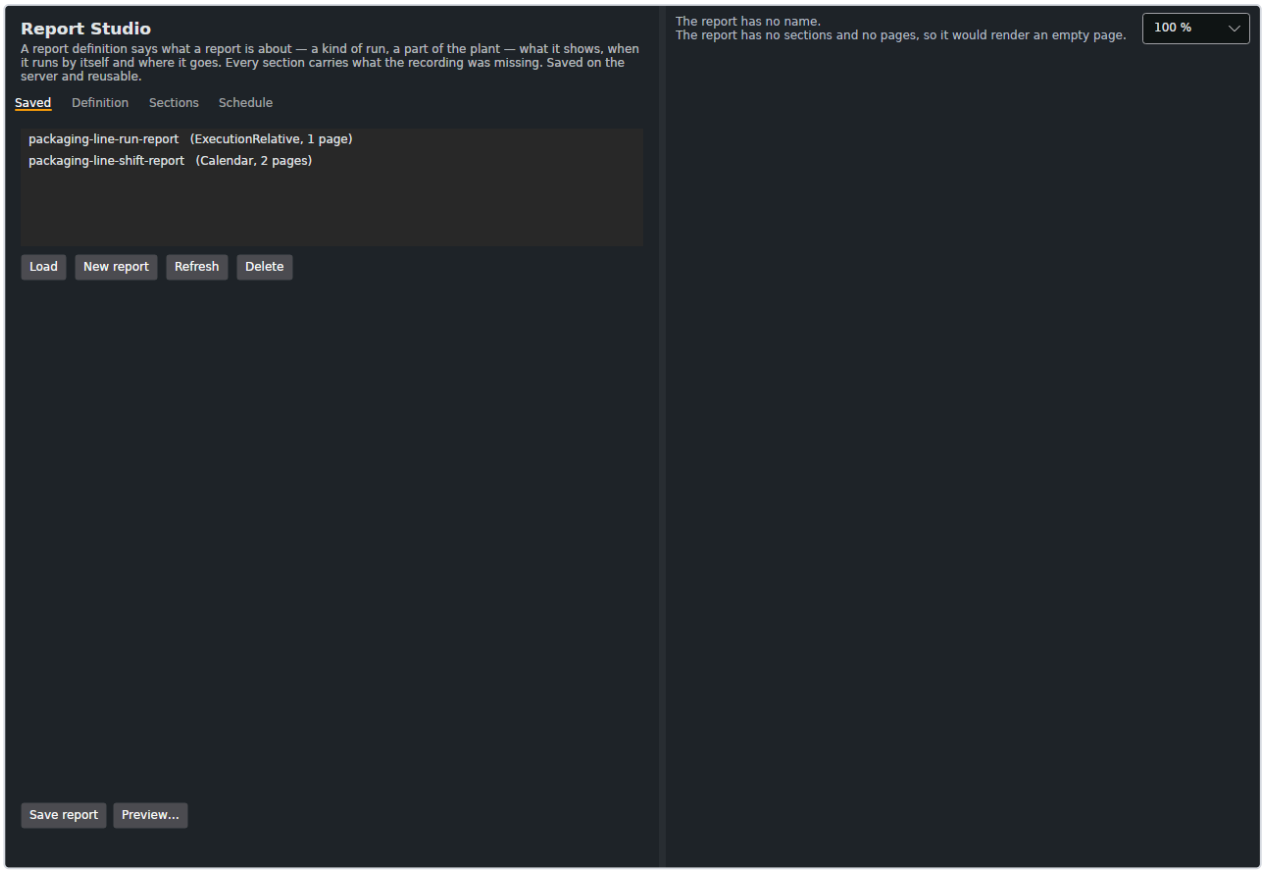
The same connection is kept for the rest of the session and is used by the Dataset Studio and the Analysis Studio as well, so you are asked at most once.

What you may do in the studio is decided by the node, by permission, and enforced there rather than in the window:

Permission	Allows
`Reports.View`	list, open and read reports
`Reports.Run`	run a report on demand, and preview one being designed
`Reports.Design`	create and edit report definitions
`Reports.Manage`	delete report definitions
`Reports.Schedule`	change when a report runs by itself, and where it is delivered

An Engineer holds all five, an Operator holds view and run, and a Viewer holds view. The separate schedule permission exists because designing a page is a change its author can undo, while turning on a schedule starts mailing the plant's numbers every morning with nobody watching.

The Report Studio window



The Desktop Report Studio with the demonstration shift report loaded: the saved reports on the left, each with its kind of window and how many pages it has, and the pages it produces drawn on the right, filled with sample rows because the Desktop previews the layout rather than running the report

The window has two halves separated by a splitter you can drag:

- **On the left, the form.** A heading and a short explanation, then four tabs — **Saved**, **Definition**, **Sections** and **Schedule** — and under them the **Save report** and **Preview…** buttons with a message line. Save sits under all the tabs because it saves the whole definition, not one tab of it.
- **On the right, the pages.** The report as it currently stands, drawn at the size it prints and redrawn about half a second after you stop typing. A zoom box at the top right offers 50 % to 200 %.

The message line under the buttons is where the studio tells you what it did ("Report saved.") and, in red, what stops a definition from being saved.

The Saved tab

The list of report definitions held on the node. Each entry shows the report's name, its kind of window and what it is built from: how many sections it has, how many pages it has if it is a page layout, or both. A report with a schedule also shows a clock symbol followed by its times of day, for example `packaging-line-shift-report (Calendar, 2 pages) ⏰ 06:00 14:00 22:00`.

Button	What it does
Load	Loads the selected report into the form for editing.
New report	Clears the form to start a new definition. The message line says "Started a new report."
Refresh	Reads the list of reports, the saved datasets and the plant's semantic model from the node again. Use it after somebody else has saved a report or a dataset.

Button	What it does
Delete	Deletes the selected report from the node. This needs `Reports.Manage`.

A report built as pages (see *Sections and page layouts* below) is counted in pages, and one built as sections in sections, for example `(Calendar, 3 sections)`. Only a report with neither shows `0 sections`, and that report really is empty.

The Definition tab: what the report is about

Field	Meaning
Name	The report's identifier on the node. Lower case with hyphens reads best, for example `packaging-line-8h-shift`. Saving under a name that already exists replaces that report.
Title	The heading printed on the report, for example `Packaging line — 8-hour shift`.
Description	One sentence saying what the report is for. Shown to people choosing a report.
Kind of window	How the report divides time. See the table below.
Type	For every kind of window except `Calendar`: which run, event or state type the report is about. The list is filled from the plant's own detection rules and follows the kind of window.
Asset (that asset and everything below it)	Narrows the report to one place in the plant's asset model, for example `Packaging/Loader`. Empty means the whole plant.
Reaches back (15m, 1h, 8h, 1d)	How far back the report reaches when it runs by itself. A scheduled shift report that runs at 06:00 with `8h` covers 22:00 to 06:00. Durations are written with a unit: `15m`, `1h`, `8h`, `1d`. A number without a unit is refused.
Most runs in one report	The most runs, events or state intervals one report covers. Default `50`. A report that reaches it says so rather than stopping quietly.

The kind of window is the same choice a dataset makes when it segments history, so a report and the datasets inside it agree about what a run is:

Kind of window	The report is about	Packaging line types
`Calendar`	a plain stretch of time — the last shift, the last day	none needed
`ExecutionRelative`	each run of a kind the plant detects	`order` (Production order), `changeover` (Changeover), `filler-run` (Pouch loader run)
`EventRelative`	each occurrence of an event type	`filler-fault`, `capper-fault`, `depalletiser-starved`, `palletiser-blocked`
`StateInterval`	each interval the plant spent in a state	`line-mode`, `filler-state`, `capper-state`

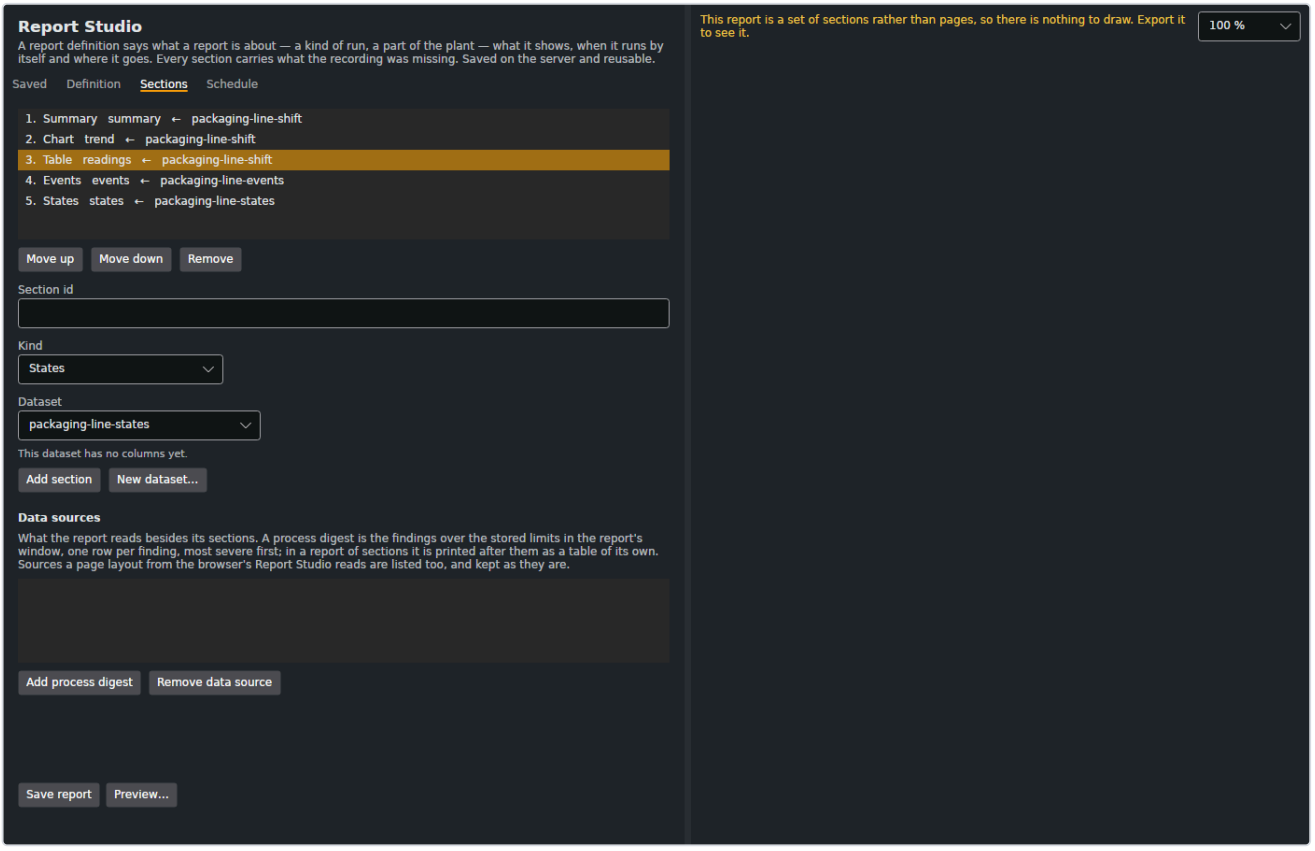
The type names come from the packaging line's detection rules (see the *Semantic model* chapter). A site that detects batches or wash cycles sees its own types here instead; nothing in the report engine knows the words "shift", "batch" or "order".

The **Type** and **Asset** boxes can also be typed into. This lets you write a report for a type whose detection rule has not been deployed to this node yet. Any kind of window other than `Calendar` must name a type: without one, the report would cover every type in the plant at once, and it is refused.

The difference between a shift report and a run report is only this tab. A shift report is `Calendar` with a lookback of `8h`; a report per production order is `ExecutionRelative` with type `order`. The sections can be identical.

The Sections tab: what the report shows

A section-based report is an ordered list of sections, read from top to bottom. The tab shows that list, numbered, with each section's kind, its id and the dataset it reads, for example `2. Chart trend ← packaging-line-shift`.



The Sections tab with five sections composed over the demonstration datasets — a summary, a chart, a table, the events and the states — in reading order, with the controls for adding another below. The preview on the right says it cannot draw a report made only of sections, as described below.

Control	What it does
Move up / **Move down**	Moves the selected section one place in the reading order.
Remove	Removes the selected section from the report.
Section id	A short identifier for a new section, unique within the report: `about`, `trend`, `rows`.
Kind	What the new section draws. See the table below.
Dataset	The saved dataset the section reads. Under it, a line reads **Draws:** followed by the columns that dataset produces, so you can see which tags a dataset carries without opening the Dataset Studio.
Add section	Adds a section with the id, kind and dataset above to the end of the list.
New dataset...	Opens the Dataset Studio. When you close it, the list of datasets is read again, so the dataset you just saved can be chosen at once.

Section kinds

Kind	What it draws	Typical dataset
`Summary`	One row per run with its aggregates: the table a run report opens with.	a run summary dataset, such as `packaging-line-runs`
`Table`	The dataset's rows as they are, for a report meant to carry the readings.	a fixed-interval dataset, such as `packaging-line-shift`
`Events`	What was raised during the report's window.	an event table, such as `packaging-line-events`
`States`	Which modes the plant was in, and for how long. Not allowed under a `StateInterval` window, where it would say nothing.	a state table, such as `packaging-line-states`
`Note`	Free text: a standing instruction, a caveat, a heading. Needs no dataset.	none
`Chart`	The section's rows drawn rather than tabulated: same dataset, same coverage, same limits.	any dataset with numeric columns

The studio refuses a section that cannot work, at the moment you add it: a section with no id, a second section with the same id, a data section with no dataset ("A section that draws data needs a dataset."), or a kind that says nothing under the chosen window ("A States section says nothing under a StateInterval scope.").

The Desktop form has no box for a note's text: a note added here is saved with empty text. Notes with text arrive with the demonstration reports and with reports made from an investigation (see the *Analysis* chapter). With the Web HMI module, the browser's Report Studio edits text on the page directly.

Chart settings

When the selected section is a `Chart` section, a **Chart** group appears under the section controls. It is hidden for other kinds, because six chart controls on a table section would say nothing about that section.

Field	Meaning
Chart type	`Line` (a line per column against the X column), `Bar` (a bar per row: what a per-run summary wants), `Scatter` (one column against another, markers only), `Histogram` (how often one column's values fell in each band), `Trend` (columns against time; a gap in recording is drawn as a gap, not joined).
X column	The column along the horizontal axis. The list offers the dataset's columns. For a trend over a fixed-interval dataset this is `timestamp`.
Columns drawn	The columns plotted, separated by commas. A scatter and a histogram take exactly one.
Left axis label	Words on the left axis, when the column names are not the right ones.
On the right scale	Columns, from those drawn, that are plotted against a second axis on the right. This is what makes a temperature and a rate readable on one chart. Not offered for a histogram.
Right axis label	Words on the right axis.
Bins	Number of bands in a histogram, 2 to 200. Default `20`. Shown only for a histogram.
Fit a line	Draws a fitted line through a scatter. Shown only for a scatter.

Field	Meaning
Apply chart	Writes these settings onto the selected section. The message line says "Chart applied to this section."

Contradictory settings are refused when you press **Apply chart**, with the reason: a scatter given two columns, a histogram given a second scale, a column on the right scale that is not one of the columns drawn.

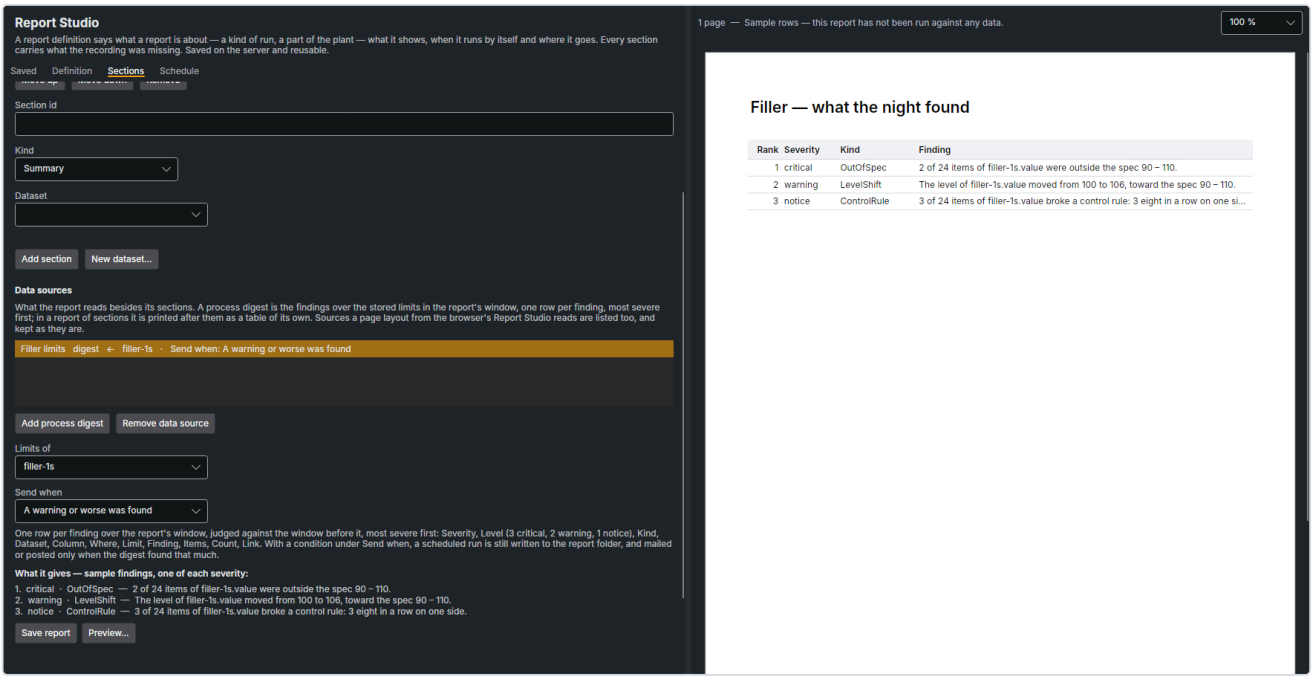
A new chart section starts with a line chart of every numeric column against the first, so it is valid the moment it is added.

Data sources: the process digest and the alarm review

Under the sections, **Data sources** lists what the report reads besides them, one line each: a process digest as its name, its id, the limits it reads and when it is sent — `Process digest digest ← Every dataset · Send when: Always` —, an alarm review as its name and its id — `Alarm performance alarms` — and the sources a page layout from the browser's Report Studio reads, which are listed and kept as they are.

Control	What it does
Add process digest	Adds a process digest and selects it: the findings over the stored spec and plausible limits in the report's window, judged against the window of the same length before it, one row per finding, most severe first — the same data source the browser's Report Studio adds (its columns are listed in the Report Studio manual, *A process digest*). A second one gets an id of its own, `digest` then another.
Add alarm performance	Adds the alarm-performance review and selects it: one row per finding of the review over the report's window, worst first, against ISA-18.2 and EEMUA 191 — `Rank`, `Verdict`, `Level` (3 unacceptable, 2 manageable, 1 acceptable, 0 for information), `Kind`, `Subject`, `Finding`, `Action`, `Magnitude`. The same data source the browser's Report Studio adds; its id is `alarms`, or another for a second one. It has nothing to set: it reads the node's alarm journal over the report's window. While it is selected, its columns and **What it gives** — the three sample findings the preview gives it, worst first — are shown under the list. Scheduled on the first of the month over the month before, it is the monthly alarm review.
Remove data source	Removes the selected source.
Limits of	*Every dataset*, or one saved dataset's limits only. Shown while a digest is selected.
Send when	*Always*; *Anything was found*; *A warning or worse was found*; *Something critical was found*. A scheduled run of a report whose digest is set to anything but *Always* is still written to the report directory, and mailed or posted only when that digest found something at least that severe; with two digests, either one reaching its own setting sends the report. A report run by hand is never held back.

Both settings are written onto the digest as you choose them. Under them, **What it gives** lists the three sample findings the preview gives a digest, one of each severity, over the dataset it names: what a table bound to it shows before the report has ever run.



The Sections tab scrolled to Data sources, with a process digest named Filler limits selected: Limits of set to the filler-1s dataset and Send when to A warning or worse was found, its three sample findings — critical, warning and notice — under What it gives, and the preview beside it printing the same three findings as the table on the report's page

In a report of sections a digest or an alarm review is printed after the sections, as a table of its own with its columns; in a page layout it is placed like any other data, by binding an element to it in the browser's Report Studio.

The Schedule tab: when it runs and where it goes

When it runs by itself

Field	Meaning
Run on a schedule	Turns the schedule on. A saved report does not run by itself unless this is ticked.
At (06:00, 14:00, 22:00)	Local times of day, 24-hour `HH:mm`, separated by commas.
Time zone (Europe/Berlin)	The plant's time zone as an IANA name. Times are local because shift boundaries are local; a schedule kept in UTC would move by an hour twice a year. A zone this machine does not recognise is refused rather than silently treated as UTC.
Format	The file a scheduled run produces: `html`, `xlsx` (Excel) or `pdf`.

Where it goes

Field	Meaning
Email to	Addresses, separated by commas or semicolons. Empty sends no mail.
Webhook URL	An endpoint the finished file is posted to. Empty posts nothing.

Under the e-mail box the window reminds you where mail goes: through the **node's mail relay**, the SMTP server, sign-in and sender set under **Acquisition ▸ Alarms… ▸ Notifications**, the same relay alarm e-mail uses. A report names only its recipients. A node with no relay still writes the report to disk and says in its log that it was not mailed. The relay is read each time a report is sent, so a relay corrected while the node runs is the one the next report goes through.

A scheduled report is produced by the node itself, whichever process holds it, and is always written to disk in the node's report directory, whether or not it is also mailed or posted.

Some consequences of how a schedule behaves:

- **Missed runs are caught up, within limits.** A node that was down for a week owes twenty-one shift reports. It produces the most recent four and records the rest as skipped, so the count tells somebody to look, rather than flooding an inbox at the moment somebody is dealing with the outage.
- **Clock changes are handled.** On the day clocks go forward, a time that does not exist (such as 02:30) runs at the instant the clock jumps to. On the day clocks go back, a time that happens twice runs only once.
- **Saving an enabled schedule with no times is refused** ("The schedule is on but names no time of day, so it would never run."), as is the same time given twice.

Changing the schedule or the delivery needs `Reports.Schedule`.

Saving

Save report checks the whole definition and, if it passes, stores it on the node and refreshes the Saved list. If anything is wrong, nothing is saved and every problem is listed in red under the button. The checks are the node's own, so a definition that saves here is one the node will run. Among them:

Problem	Fix
"The report has no name."	Fill in **Name**.
"The report has no sections and no pages, so it would render an empty page."	Add at least one section.
"'8' is not a duration. Use 15m, 1h, 8h or 1d."	Give **Reaches back** a unit.
"A ExecutionRelative report has no type selected, so it would cover every type in the plant."	Choose a **Type**.
"'06.00' is not a time of day. Use 06:00."	Write times as `HH:mm`.
"'Europe/Berln' is not a time zone this machine knows."	Correct the time zone name.
"Section 'trend' draws data but binds no dataset."	Remove the section and add it again with a dataset.

Previewing

The pages on the right of the window, and the separate window **Preview…** opens, show what the report will look like on paper. The pages are laid out in millimetres by the same layout engine the node uses and drawn from the same boxes the PDF renderer draws, so what the preview shows and what comes out of the printer are one geometry. Use the zoom box to look closely.

Preview… opens the same drawing in its own window, which is useful on a small screen. Neither saves anything.

The preview covers the last eight hours. Where the node's data is reachable it is used. Where it is not — no plant running, or a dataset the node cannot read — the layout is drawn over **sample rows**: a visible ramp over round numbers, never anything that reads like a measurement, and both the status line and

the report's own coverage line say so. The questions a preview answers are mostly about layout (does the table fit, does a long heading wrap and push the totals onto page two), and those have an answer before the line is running.

A definition that cannot be built yet shows, where the pages would be, what is still missing. This is the ordinary state of a form being filled in, not an error.

Sections and page layouts

A report can be described in two ways:

- as **sections**, an ordered list, which is what the Desktop form edits; or
- as **pages**, elements placed on paper in millimetres, with headers, footers, bands and page numbers.

The demonstration reports, and reports converted in the browser, are page layouts. When you load one here, its **Sections** tab is empty and its pages appear on the right; you can still change its name, title, scope, schedule and delivery, and saving keeps its pages. A report may not carry both descriptions at once, so sections cannot be added to a page report from the Desktop form.

A report made only of sections has no page layout to draw: the preview says "This report is a set of sections rather than pages, so there is nothing to draw." It is still a complete report. The node renders it section by section when it runs, to HTML, Excel or PDF, exactly as it always has.

With the Web HMI module, reports can also be designed and viewed in the browser: a visual Report Studio places elements on the page, converts a section report into pages with one button, and runs any saved report on demand with a chosen window and parameters, to read on screen or download as PDF or Excel.

Running a report and getting the file

The Desktop Report Studio defines, previews and schedules reports; it does not run a saved report on demand. A report reaches its readers in these ways:

1. **On its schedule.** The node runs it at each time of day in the chosen format and writes the file to its report directory, then mails it and posts it if **Email to** and **Webhook URL** say so.
2. **On demand, in the browser** (with the Web HMI module): choose the report, the window and any parameters, read it on screen, or download it as PDF or Excel.
3. **Through the node's data-access API**, by a tool of your own.

The three formats contain the same report:

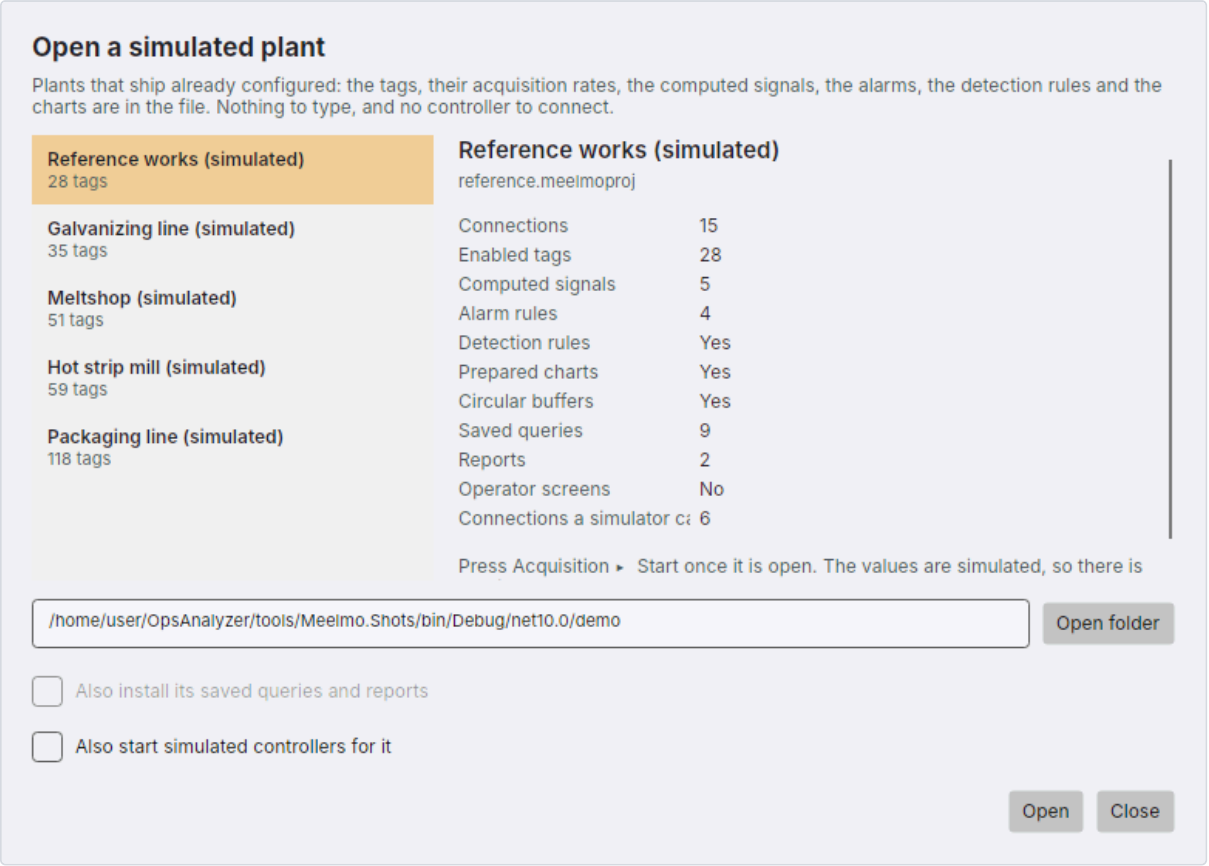
Format	Best for
`html`	reading on screen and in mail; uses the reader's own fonts
`pdf`	printing and archiving; pages exactly as previewed
`xlsx`	taking the tables into a spreadsheet

If a PDF is refused with a message about "empty boxes", the report contains characters the embedded font cannot draw; choose `html` or `xlsx` for that report.

Two other outputs in the Desktop are not report definitions and should not be confused with them. **File ▸ Export Report (HTML)…** writes a one-off HTML page of the signals currently plotted in the trend graphs (see the *Trends* chapter). **Save report…** in the Recording Timing Analysis, Diagnostics and Commissioning windows saves those windows' own findings.

The demonstration reports

A saved dataset and a report belong to the node, not to the project, so they cannot travel inside a `.meelmoproj` file. **Help ▸ Demonstration…** installs them for you: choose **Packaging line (simulated)**, tick **Also install its saved queries and reports**, and press **Open**. The status line then says how many were installed and how many were already there.



The demonstration dialog, listing the simulated plants and how many saved queries and reports each installs

Nothing is overwritten: a definition whose name the node already holds is left as it is. On a workstation that is a client of a node elsewhere, the option is unavailable, because those definitions are that node's and changing them is its administrator's decision.

For the packaging line, the definitions are named after the plant (`packaging-line-…`).

The saved datasets

Dataset	What it holds
`packaging-line-shift`	Eight charted signals at one-minute cadence, good-quality readings only, each gap longer than five minutes left empty, with a 15-minute rolling average of the first column beside it. The rows a trend and a readings table are drawn from.
`packaging-line-raw`	Every sample of two signals exactly as recorded: one row per sample, nothing resampled.
`packaging-line-events`	The events the detection rules raised.
`packaging-line-states`	The states the plant was in and for how long.

Dataset	What it holds
`packaging-line-runs`	One row per production order (`order`), with each signal's average, spread, extremes and change over the run.
`packaging-line-cycles`	The same orders as overlaid curves.
`packaging-line-event-window`	A window around each occurrence of the first declared event (`filler-fault`).
`packaging-line-by-state`	The measurements cut by the first declared state (`line-mode`).
`packaging-line-synchronised`	A batch-sampled table over the project's temporal coherence frame.

The eight columns of `packaging-line-shift` are the signals the demonstration's charts open on, measured signals first and then two computed ones: `MagazineLevel` , `Erector Rate` , `BlanksPerMinute` , `BottomSealer Rate` , `SealPressure` , `SealTemperature` , `Availability` and `OEE` . A column is qualified with its machine only where the tag name alone would be ambiguous, which is why the two rates read `Erector Rate` and `BottomSealer Rate` .

The shift report

`packaging-line-shift-report` , titled **Packaging line (simulated) — shift report**. Its scope is `Calendar` , reaching back `1h` : an hour of the plant, so that it has something to show soon after the demonstration starts. It contains:

1. **About this report** — a note: the last hour at one-minute cadence, bad-quality values excluded, gaps longer than five minutes left empty.
2. **How it ran** — a trend of up to four numeric columns of `packaging-line-shift` against `timestamp` .
3. **Readings** — the rows of `packaging-line-shift` , at most 500.
4. **What was raised** — the events from `packaging-line-events` , at most 200.
5. **Where the time went** — the states from `packaging-line-states` , at most 200.

The run report

`packaging-line-run-report` , titled **Packaging line (simulated) — run report**. The same engine with one field different: its scope is `ExecutionRelative` on type `order` , reaching back `24h` , so its rows are production orders rather than minutes. It contains a note explaining what a run is, a **Runs** summary of `packaging-line-runs` , and **Run by run**, a bar per order. Bars rather than a line, because fifty runs are fifty separate things and a line would suggest a continuity they do not have.

Both reports arrive as page layouts, so they open with pages on the right of the studio rather than an empty preview. Read them, copy them under a new name, or delete them.

Worked example: an 8-hour shift report for the packaging line

The demonstration shift report covers one hour. The packaging line runs three eight-hour shifts, starting at 06:00, 14:00 and 22:00, and the shift leaders want a PDF of the shift that has just ended, in their inbox at each handover.

Before you start: open the packaging demonstration with its saved queries and reports installed (see above), start acquisition with **Acquisition ▸ Start**, and let it run for a while so the historian holds something to report on. Sign in as a user who holds `Reports.Design` and `Reports.Schedule` , such as `admin` .

1. Choose **Data ▸ Report Studio…**. On the **Saved** tab, select `packaging-line-shift-report` and press **Load**. The pages appear on the right.
2. On the **Definition** tab, change **Name** to `packaging-line-8h-shift`, so the original stays as it is.
3. Change **Title** to `Packaging line — 8-hour shift` and **Description** to `The shift that has just ended: trend, readings, events and line states.`
4. Leave **Kind of window** at `Calendar`, **Type** empty and **Asset** empty: the report is about the whole line.
5. Change **Reaches back** from `1h` to `8h`. Leave **Most runs in one report** at `50`.
6. Watch the pages on the right redraw. The readings table is now longer: eight hours at one-minute cadence is 480 rows, and the table continues onto further pages with its column headings repeated. Set the zoom to `75%` to see several pages.
7. On the **Schedule** tab, tick **Run on a schedule**. In **At** type `06:00, 14:00, 22:00`. In **Time zone** type the plant's zone, for example `Europe/Berlin`. Set **Format** to `pdf`.
8. In **Email to** type the shift leaders' addresses, for example `shift-a@example.com; shift-b@example.com; shift-c@example.com`. Leave **Webhook URL** empty.
9. Press **Save report**. The message line reads "Report saved." and the Saved list now shows `packaging-line-8h-shift (Calendar, 2 pages) ⏰ 06:00 14:00 22:00`.
10. Check that the node has a mail relay: **Acquisition ▸ Alarms… ▸ Notifications**. If it has none, the report is still written to disk at each handover and the node's log says it was not mailed.

At 06:00 the node composes the report over 22:00–06:00, writes `packaging-line-8h-shift` as a PDF to its report directory, and mails it. The last page carries the coverage line: if the line was stopped for part of the night, the report says how many readings were missing rather than averaging over what it had.

A second example: a checkweigher section report

Here the report is built from sections, over a dataset of its own.

1. In the Report Studio press **New report**. Name it `packaging-checkweigher`, title `Checkweigher — case weight`, **Kind of window** `Calendar`, **Reaches back** `8h`.
2. On the **Sections** tab press **New dataset…**. In the Dataset Studio create a wide, fixed-interval dataset named `packaging-checkweigher-1m` with the signals `Packaging/Checkweigher/CaseWeight`, `Packaging/Checkweigher/WeightTarget` and `Packaging/Checkweigher/Rate` at an interval of one minute, and save it (see the *Datasets* chapter). Close the Dataset Studio.
3. Back in the Report Studio, type `weights` in **Section id**, choose **Kind** `Chart` and **Dataset** `packaging-checkweigher-1m`. The **Draws:** line lists the dataset's columns. Press **Add section**.
4. Select `1. Chart weights` in the list. In the **Chart** group choose **Chart type** `Trend`, **X column** `timestamp`, and in **Columns drawn** type the case-weight, weight-target and rate columns as the **Draws:** line names them, separated by commas. Put the rate column in **On the right scale**, type `Weight` in **Left axis label** and `Rate` in **Right axis label**, and press **Apply chart**.
5. Add a second section: **Section id** `spread`, **Kind** `Chart`, the same dataset, **Add section**. Select it, choose **Chart type** `Histogram`, put only the case-weight column in **Columns drawn**, set **Bins** to `30`, and press **Apply chart**. This shows how case weights were distributed around the target over the shift.
6. Add a third section: **Section id** `rows`, **Kind** `Table`, the same dataset.
7. Add a fourth: **Section id** `raised`, **Kind** `Events`, **Dataset** `packaging-line-events`.
8. Use **Move up** and **Move down** until the order is `weights`, `spread`, `raised`, `rows`: pictures first, the rows behind them last.
9. Press **Save report**.

Because this report is made of sections, the preview on the right says it is a set of sections rather than pages. It runs on a schedule like any other; tick **Run on a schedule** on the **Schedule** tab to have it produced at each handover. With the Web HMI module it can also be run in the browser straight away, or converted there into a page layout to adjust its look.

Reports made from an investigation

Make a report in the Analysis Studio writes an investigation out as an ordinary saved report named `investigation-` followed by the date and time of the occurrence, for example `investigation-20260922-1432`. It opens in the Report Studio like any other report and can be printed, attached to a handover or read again later. The *Analysis* chapter describes what it contains.

Limits

A report is bounded at every step, so a schedule cannot turn into a request for millions of rows against the plant server:

- each section has a row ceiling (the demonstration uses 500 for readings and 200 for events and states), and the host's own maximum rows per query applies as well;
- the scope has a maximum number of runs (**Most runs in one report**, default 50);
- a run is bounded by the host's query timeout and can be cancelled.

A table that would exceed its ceiling is cut and says so.

When something is wrong

The Saved list is empty and a message names a connection problem. The studio could not reach the node's data-access service, or the token was refused. Close the studio and open it again to be asked for the address and token.

The Type list is empty. The kind of window is `Calendar`, which needs no type, or the node has no detection rules yet. You can type a type name by hand.

A figure prints an em dash. The column resolved to nothing: a field the dataset does not produce, or a window with no readings. Check the **Draws:** line for the column names and preview over a period you know has data.

The chart is empty. A chart needs an X column and at least one column drawn, and they must not be the same column.

Saving the schedule is refused. Changing the schedule or delivery needs `Reports.Schedule`. Ask an administrator to grant it at the node.

A scheduled report was written but not mailed. The node has no mail relay, or the relay refused the message. See **Acquisition ▸ Alarms… ▸ Notifications** and the node's log.

Analysis

Trends show what happened to the signals you chose to look at. The tools in this chapter are for the harder question: *something went wrong, and nobody knows where to look*. They read the historian, compare every signal the project publishes against its own normal behaviour, and narrow four hundred signals to the half-dozen that were doing something different, in the order they did it.

This chapter covers:

- which tool answers which question;
- run comparison, for runs that did not go like the good ones;
- the **Analysis Studio** in the Desktop: finding the moment, ranking what changed, finding similar windows, remembering a fault as a rule, checking whether it has happened before, and the correlation study;
- the **Recording Timing Analysis**, briefly;
- a worked example on the packaging line.

Everything here reads the historian, so the historian must be configured and must have recorded the period you ask about (see the *Historian* chapter). The run, event and state types the tools offer come from the plant's detection rules (see the *Semantic model* chapter).

Fitted models — soft sensors, forecasts and the residual watch — are not part of the base product. They belong to the Models module, which has its own chapter.

Which tool, from where you are standing

The tools look alike from a distance: they all read history, all produce a ranked list, and all end with a sentence saying the list is not a verdict. They are not alike. Each answers a different starting position, and using the wrong one gives a confident answer to a question nobody asked. Start from what you already know.

You have	You want	Use
Which signals matter, and which runs were good	Why *this* run was different from the good ones	**Run comparison**
A complaint, but no time ("nights were worse")	The moment it happened	**Analysis Studio ▸ Find moments**
A moment (an event, an alarm, a time somebody noted)	Which signals were doing something different, and in what order	**Analysis Studio ▸ Investigate**
An investigated window	The other times the plant looked like that	**Find windows like this**
An investigation you understand	The plant to watch for it from now on	**Remember this fault…**
A rule, new or old	What it would have said over the last months	**Has this happened before?**
A unit and its tags, nothing else	Which of them relate to each other at all	**What relates to what**
A failure you anticipated well enough to write a rule for	To be told when it happens again	an advisory rule (see the *Semantic model* chapter)

The usual investigation runs down the middle of that table:

1. *Something was off on nights.* **Find moments** sweeps the period and returns the moments at which several signals changed level together.
2. *Investigate the 02:14 moment.* The ranking says which signals left their normal band, and which moved first.
3. *Has this happened before?* **Find windows like this** searches for the same shape, or **Remember this fault…** turns the finding into a rule and replays it over 90 days.
4. *Arm it.* The rule is pasted into the project's detection rules, reviewed, and activated like any other change.

What every tool refuses to tell you

Each tool deliberately stops short of a conclusion, and the refusal is part of the design:

Tool	Refuses to	Because
Find moments	report drift or cycling as a moment	It looks for a level that jumped **once and stayed**. A signal that cycles or drifts is a different shape and is set aside by name.
Investigate	call anything a cause	Signals driven by one upstream event all depart, in some order, and none of them caused the others.
Find windows like this	search from a flat window	Scaling a window with no movement by its own spread turns instrument noise into a pattern that matches every quiet window in the year.
Remember this fault	be trusted on one example	A threshold from a single occurrence either never fires again or fires constantly, and neither is visible by reading it. That is what the replay is for.
What relates to what	report a strong correlation as a relationship	Slow signals carry far less evidence than their sample count, and a large study produces strong correlations by chance.

Everything these tools compute is arithmetic you can check by hand: means, standard deviations, thresholds, dwell times and correlations. Nothing is trained or predicted, so the same question over the same history gives the same answer twice.

Run comparison

Run comparison answers *why was this run different?* You nominate the runs that went well as references, and the runs to judge against them. The runs are aligned on **phase, not wall-clock time**: each run is measured as a fraction of its own length, or by time elapsed since it started. A reference band of mean ± σ is drawn from the good runs, and each signal of the run being judged is reported with how far it left the band and how much of it was covered by recorded data.

Aligning on phase is what makes this useful. Two production orders that took different times are still comparable stage by stage, where wall-clock alignment would make every one of them look wrong.

On the packaging line the runs are the types the detection rules declare: production orders (`order`), changeovers (`changeover`) and pouch loader runs (`filler-run`). A typical question is why order `B-2291` ran slower than the last five: tick the five as references and `B-2291` as the run to compare, choose `Packaging/Loader/Rate` and `Packaging/Loader/CollatorBacklog`, and read where along the order each left the band.

Run comparison needs at least two completed runs of the chosen type in the period. A run still in progress is marked and left out of the band, and the coverage beside each deviation says how much data the comparison actually had — a small deviation and a comparison with almost nothing to compare look the

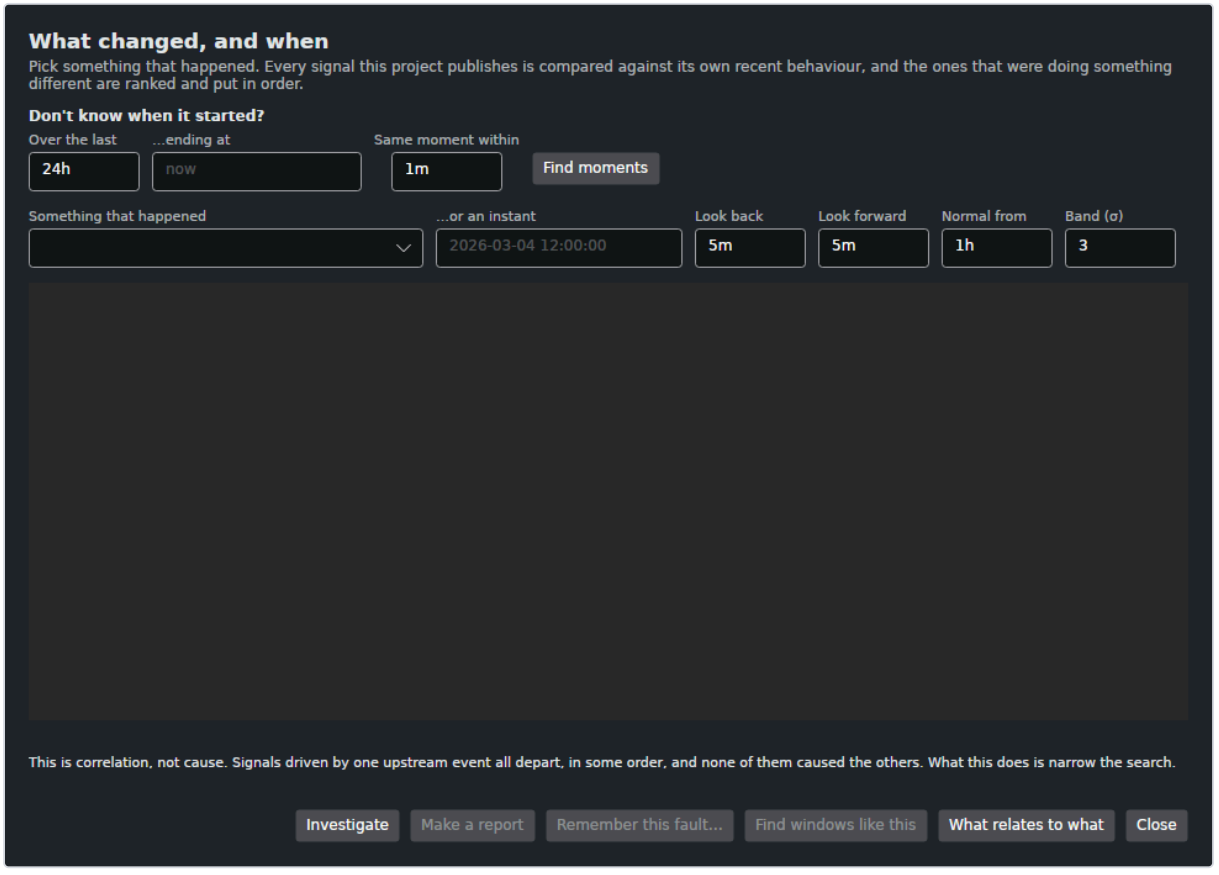
same without it.

Run comparison is provided by the node's analysis service. With the Web HMI module it is available in the browser as **Analysis ▸ Compare runs**, which draws the reference band and the judged runs together with the deviation table.

The Analysis Studio

Data ▸ Analysis Studio… opens the studio. Like the Report Studio, it is a client of the node's data-access service: on a workstation that hosts its node, or one attached to a node, it connects without asking; otherwise it asks once per session for the server address and an API token (see the *Reports* chapter).

The studio compares **every signal the open project publishes**. Open the project for the plant you want to investigate before opening the studio; a project that publishes no signals is refused with "This project publishes no signals to compare."



The Analysis Studio on the packaging line after Find moments and Investigate: the moment chosen at the top (23 signals moved together; its strength shown as ∞ σ because some of them had never varied before), and 86 of 146 signals ranked by when they stepped, from the buffer between the Loader and the Checkweigher five minutes before to the Checkweigher faulting a minute and a half after

The window is titled **What changed, and when**, and reads from top to bottom:

1. **Don't know when it started?** — the sweep, which finds moments for you.
2. The occurrence to investigate, and the settings for the investigation.
3. The ranked list of signals.
4. Three lines of text: the status of the last action, the caveat, and a footnote about what the last search can and cannot see.
5. The buttons: **Investigate**, **Make a report**, **Remember this fault…**, **Find windows like this**, **What relates to what** and **Close**.

The fields

Field	Default	Meaning
Over the last	`24h`	The period the sweep, the shape search and the correlation study cover, written as `8h`, `24h`, `7d`.
…ending at	now	The end of that period. Leave empty for now, or type a date and time such as `2026-09-22 06:00:00`.
Same moment within	`1m`	How close in time changes must be for the sweep to call them one moment.
Something that happened	—	A list of occurrences to investigate: events the detection rules raised in the last week, moments the sweep found, and windows the shape search found. Choosing one copies its time into **…or an instant**.
…or an instant	—	The time to investigate, typed as `2026-09-22 14:32:41`, when it is not in the list.
Look back	`5m`	How far before the instant the investigated window starts.
Look forward	`5m`	How far after the instant the window ends.
Normal from	`1h`	How much history before the window is read as each signal's normal behaviour (its baseline).
Band (σ)	`3`	How many standard deviations from its baseline a signal must go, and stay, to count as having left.

When the studio opens, it lists the events the node detected in the last seven days, newest first, for example `2026-09-22 14:32:41 filler-fault (Packaging/Loader)`. If there are none, the status line says "Nothing detected in the last week. Type an instant instead."

Don't know when it started? — Find moments

The ordinary complaint is not "at 14:32 the line stopped". It is that today was worse than yesterday and nobody can say when it turned. **Find moments** is for that.

1. In **Over the last** give the period, for example `8h` for the last shift.
2. Optionally give **…ending at**, to sweep a shift in the past.
3. Leave **Same moment within** at `1m`, or see below.
4. Press **Find moments**.

The sweep walks every signal in the project looking for a level that **jumped once and stayed**, then groups changes that fell within **Same moment within** of each other. Each moment found is added to the top of **Something that happened**, labelled with its time, how many signals moved together and the strongest shift among them:

```
2026-09-22 02:14:05  ·  5 signals moved together (7σ)
```

The first one is selected for you. The status line says how many moments were found from how many signals, and how precisely a moment can be placed — the sweep reads bucketed history, so it says *within a minute* rather than pretending to a second. Investigating a moment then reads the raw samples and says when each signal moved.

Two things to know about the answer:

- **It ranks by how many signals moved, not by how far.** The biggest single step in a day is usually a setpoint somebody typed. Five signals moving inside a minute is the plant.

- **An empty list is not a quiet day.** This finds steps. A signal cycling evenly through the period produces nothing here; one whose cycle does not close, or one drifting steadily, produces more changes than a moment can explain and is **set aside and named in the status line**. Read that line. A signal named there is doing something — just not the thing this looks for.

Same moment within is the one setting with no principled value. A pressure and the level it feeds move together within seconds; a machine and the buffer downstream of it within minutes. Too small and one event becomes several moments; too large and an hour of unrelated activity becomes one. `1m` suits the packaging line, where the machines are coupled by short conveyors.

Investigate: what changed

1. Choose an occurrence in **Something that happened**, or type an instant in **…or an instant**.
2. Adjust **Look back** and **Look forward** to cover the disturbance. `5m` either side suits a fault that trips a machine; a slow deterioration needs longer.
3. Leave **Normal from** at `1h` unless the hour before the window was itself unusual — a changeover, for example. The baseline must be a period of normal running.
4. Press **Investigate**. The status line shows "Reading…" and then, for example, "4 of 118 signals were doing something different."

Every signal is compared against **its own recent behaviour**, not against a configured limit. The baseline gives each signal a mean and a standard deviation. A signal that passes the band **and stays past it** has left; the dwell is there for the same reason an alarm has a raise delay, because a single sample beyond three sigma is what three sigma means. That difference is the point: a buffer that normally sits at 40 % and is at 55 % has departed even though no alarm limit was crossed, and that is usually the interesting one.

Reading the ranking

Each row is one signal:

Column	Meaning
Signal	The signal id, in bold if the signal departed.
Verdict	What it did — see below. A word rather than a colour, so it reads correctly for everybody.
Peak	How far it went, in standard deviations of its own baseline, for example `7.2σ`. A step change shows `—`, because its baseline had no spread to measure against.
Value	The peak value, with the signal's unit.
When	How long **before** or **after** the occurrence it started to move, for example `40 s before` or `1.2 min after`.

The verdicts:

Verdict	Meaning
Left its band	Passed the band and stayed past it.
Step change	Held one value through the whole baseline and then moved — typically a boolean such as `Faulted` going true. Reported separately, because a σ over a baseline with no spread is meaningless, and a boolean going true is often the most informative thing in the window.
Steady	Stayed within its band. A signal that spiked once shows its peak but is not called a departure, so you can see and argue with the dwell.

Verdict	Meaning
No baseline	There was no history before the window to judge it against. "No baseline" is a verdict, not the same as steady.
No data	Nothing was recorded for it in the window.

Read the When column first. A signal that moved four minutes *before* the occurrence is a candidate; one that moved two minutes *after* it is a consequence. The order is most of the value.

Samples the driver marked as bad quality are not believed — they do not move a mean — but they are **counted** and the count is reported, because a transmitter that dropped out is frequently the whole answer. A signal too fast to read whole within the window is judged from buckets that keep each bucket's minimum and maximum, so an excursion keeps its size and its moment; its departure is then placed to within one bucket. Anything the investigation could not account for — signals left out because your account may not read them, for example — is appended to the status line.

If nothing left its band the status line says so: "No signal left its own band. That is a finding: whatever happened is not visible in these signals, and the next question is which ones are missing."

> **This is correlation, not cause, and the window says so on its face.** Signals driven by one upstream event all depart, in some order, and none of them caused the others. What this does is turn four hundred signals into six. The reasoning is yours.

Make a report

Make a report writes the finding out as an ordinary saved report, so it survives being handed to somebody else. The status line says, for example, "Saved as 'investigation-20260922-1432'. Open it in the Report Studio."

The report is built in this order, deliberately with the caveat first:

1. **What this is** — the occurrence, the window examined, the baseline period, the band, and the caveat.
2. **What moved, and when** — the sequence, earliest first, each with its distance from the occurrence.
3. A trend of the signals that moved, around the window.
4. **Every signal asked about** — the ranking, up to 20 rows.
5. **What this could not account for** — the warnings, if there were any.
6. **The readings** — the window's own readings, up to 500 rows, so a reader can check the shape rather than take the drawing on trust.

It is a normal report: open it in the Report Studio, print it, attach it to a handover, or read it again when the same thing happens (see the *Reports* chapter). Making a report needs `Reports.Design`.

Find windows like this

Find windows like this answers *has the plant done this before?* without anything having to be named. It is enabled once an investigation has found signals that moved.

It takes the window you just investigated, makes a shape from the signals the ranking says moved (up to six, in the order they moved), and searches the period in **Over the last** for other windows that resemble it. The signals come from the ranking, not from the whole project: the investigation has just worked out which six of four hundred were doing something, and searching on the others would drown the shape in signals that sat still.

It matches by **shape, not level**. Each window is scaled by its own spread first, so the same excursion at four bar and at six bar is the same excursion. Overlapping matches are suppressed, so one recurrence gives one answer rather than forty near-identical ones.

The matches go into **Something that happened**, closest first, each with its distance:

```
2026-09-21 03:02:10  ·  looks like this one (0.41)
```

Pick one and press **Investigate** to see whether it really was the same thing. The status line says how many were found, or "No other window over this period looks like it. That is a finding in itself — whatever happened, the plant has not done it again here."

Read the distances by **order and gap**, not by value. A distance is a root-mean-square difference in standard deviations and has no absolute meaning: `0.4` is not "40 % different", and there is no threshold below which two windows are "the same". If the best three are far closer than everything after them, those three are a family. If every distance is about the same, nothing resembles your window more than anything else does.

> **A resemblance is not a recurrence.** Two windows looking alike is a reason to go and look at the second one, not evidence that the same thing happened.

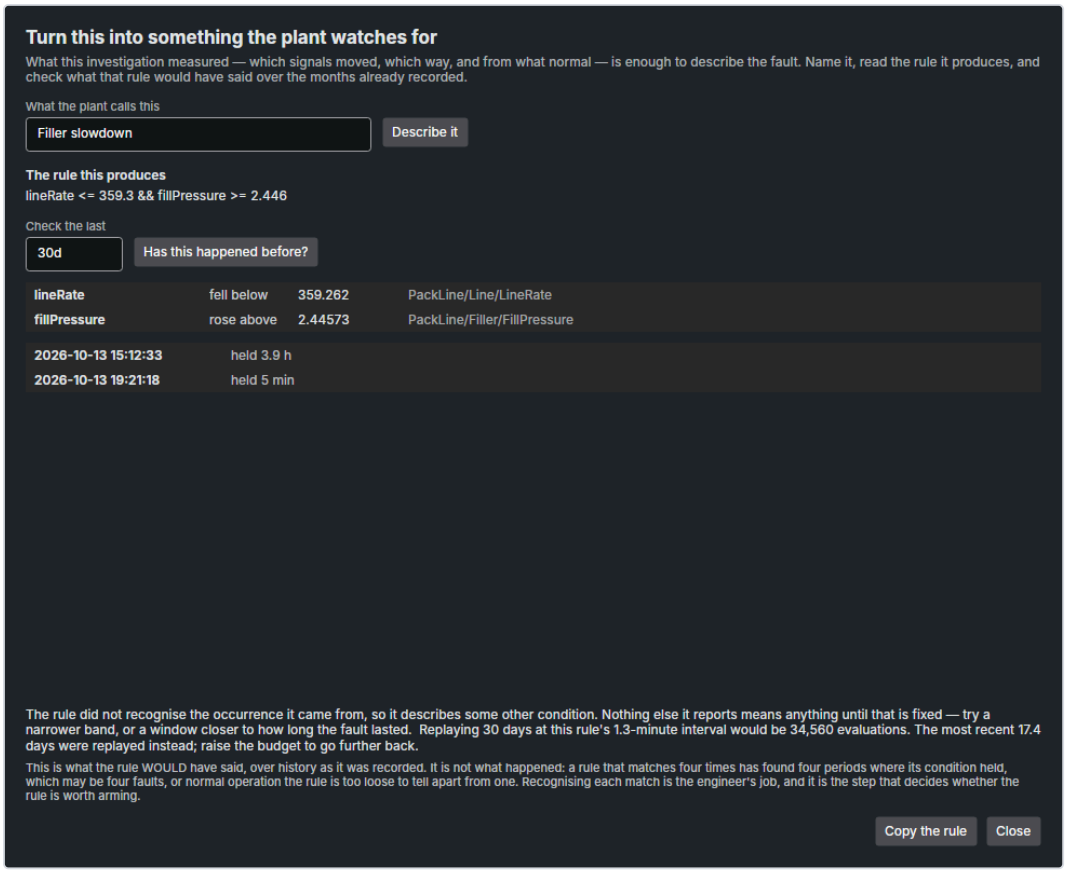
A window in which nothing moved is refused, and so is a shape that includes a signal your account may not read: searching without it would be searching for a different shape.

To search further back, set **Over the last** to `7d` or `30d` before pressing the button.

Remember this fault…

A report survives being handed to somebody; it does not survive being forgotten. And an advisory rule, which is what actually watches a plant, needs somebody to have anticipated the failure well enough to write one. **Remember this fault…** joins the two: the investigation has already measured everything a rule needs — which signals moved, which way, and from what normal — so the first occurrence writes the rule.

It is enabled once an investigation has found signals that moved, and opens a window titled **Turn this into something the plant watches for**.



Remember this fault: an investigation named, compiled into a rule in the plant's own expression language, and replayed over the last 30 days

The figure was captured on a small filler test project; on the packaging line the signal names are those of the packaging line.

Work through it in order:

1. **Name it.** Type what the plant calls this fault in **What the plant calls this**, for example `Loader starved of pouches`. A name is required — "A rule nobody named is one nobody recognises in a list a year from now." The name becomes the rule's id and the sentence an operator reads.
2. **Press Describe it.** Under **The rule this produces** the window shows the rule in the same expression language as alarms and virtual signals, for example `rate <= 38.2 && collatorBacklog >= 61.5`. Below it, one row per signal says what it did (**rose above** or **fell below**), the threshold and the full signal id.
3. **Read the rule.** Check it at a glance. A generated rule taken on trust is exactly what this is trying not to be.
4. **Check it.** Set **Check the last** (default `90d`; forms such as `30d` or `6mo`) and press **Has this happened before?** — see the next section.
5. **Copy the rule** puts it on the clipboard as a rules document, once it has been described.

Where the thresholds come from: a signal that **left its band** is cut at the edge of that band — the same line the investigation used to call it a departure. A signal that **stepped** has no band, so it is cut halfway between the level it held and the level it went to, which gives a boolean going true the threshold `0.5`. Both are loose on purpose: a threshold fitted to one occurrence's peak would be silent next time, and silence is the failure nobody can see. Too loose announces itself, and tightening from there is a decision made against data.

The window reminds you under the lists: "This was taken from one occurrence. A threshold derived from a single event either never fires again or fires constantly, and neither can be seen by reading it — which is what the replay is for."

Has this happened before?

The rule is replayed over the period in **Check the last**, and every time it would have spoken is listed with its date and how long it held (`held 3.9 h`, or `still holding`). The occurrence the rule was taken from is in the list too, marked **this occurrence**. It is not counted as evidence — it is where the numbers came from — but it is shown, because a rule that does not recognise its own source is broken.

The replay is the real rule engine, not a simulation of it: it builds an advisory engine, steps its clock by hand and runs the same evaluation the node runs, so confirmation and re-arm apply exactly as they would if you armed it.

Read the answer like this:

What comes back	What it means	What to do
"The rule did not recognise the occurrence it came from..."	The rule describes some other condition. Nothing else it reports means anything.	Investigate again with a narrower band (a lower **Band (σ)**) or with **Look back** / **Look forward** closer to how long the fault lasted, and describe it again.
"Not once in the 90 days replayed."	A first occurrence.	Worth knowing, and it makes the rule worth arming.
A handful of dates	The fault may be older than you thought.	Look at each date. The ones you recognise are the evidence; the ones you do not mean the rule is catching something else too.
Twenty or more	The rule describes normal running. Armed, it would fill the advisory strip until nobody read it.	Tighten a threshold and check again.

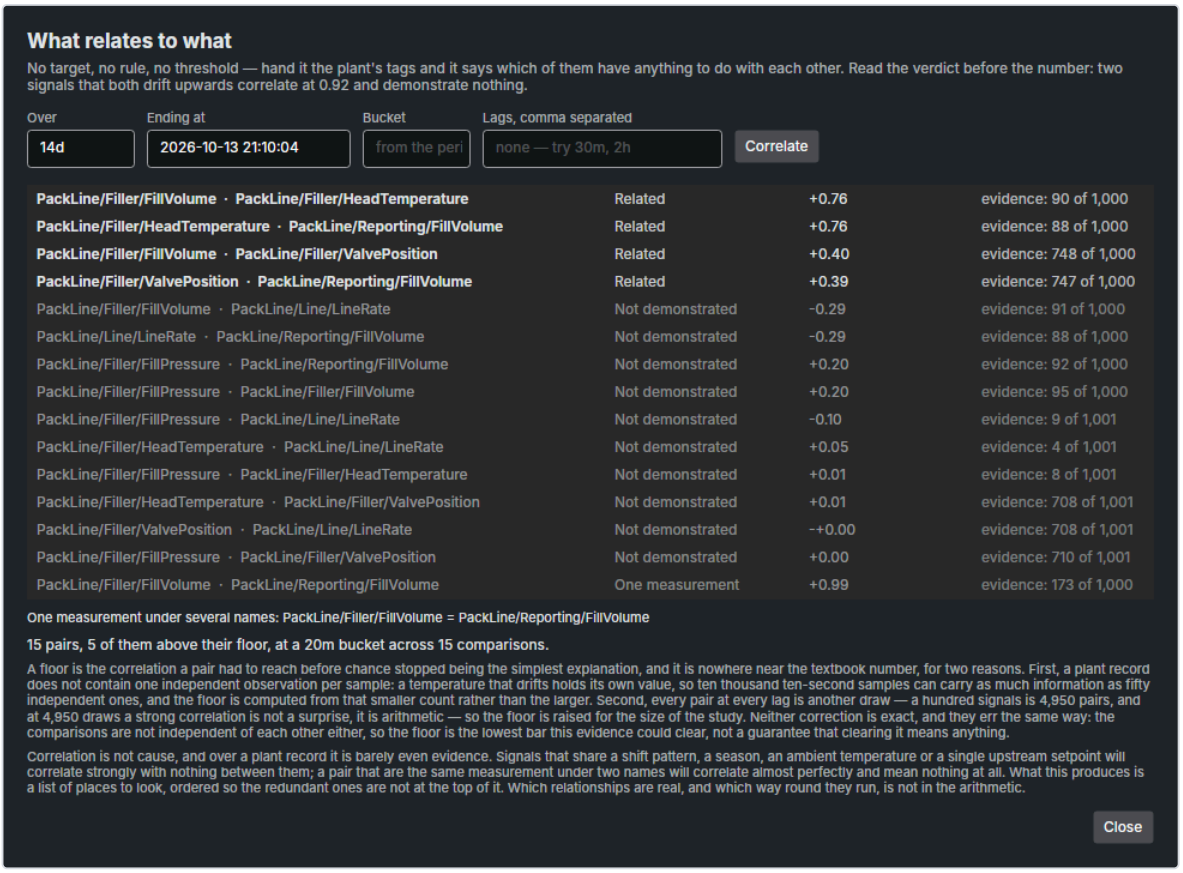
A long replay is bounded by an evaluation budget. The rule is evaluated at its own interval, so what gives is the range rather than the fidelity, and the answer says how many days it actually covered.

> **A replay is what the rule would have said, not what happened.** Four matches are four periods in which a condition held. Recognising them is the engineer's job, and it is the step that decides whether the rule is worth arming.

Arming the rule. Nothing in this window arms anything. Press **Copy the rule**, open **Acquisition ▸ Detection Rules…**, paste it in, review the change, and activate the revision — the same path as a rule written by hand (see the *Semantic model* chapter). The distance between "here is a rule" and "this plant is running it" stays exactly one person.

What relates to what

What relates to what is the one analysis that needs nothing named: no occurrence, no target, no threshold. It is enabled from the moment the studio opens and asks which of the project's signals have anything to do with each other.



What relates to what: every pair of signals with its verdict, its correlation, and how much independent evidence it rests on

The figure was captured on a small filler test project.

Field	Meaning
Over	The period studied. It starts from the studio's **Over the last**, for example `24h`; `14d` gives a fortnight.
Ending at	The end of the period, filled in from the studio.
Bucket	The interval the signals are averaged into before they are compared. Empty derives one from the period.
Lags, comma separated	Optional delays to try, such as `30m, 2h`. Each is tried in both directions, so a pair comes back in the order the arithmetic found — which signal moved first, and by how long.
Correlate	Runs the study.

Each row is one pair of signals: the two names (joined by `·`, or by `→` when the first leads the second, with the lag beside it), the verdict, the correlation `r`, and the **evidence**, for example `evidence: 729 of 1,000`.

Verdict	Meaning
Related	The correlation clears the pair's floor: chance is no longer the simplest explanation.
Not demonstrated	It does not. However strong the number looks, the evidence does not support it.
One measurement	The two are one measurement published under two names (for example a raw tag and its reported copy). Listed under the table as "One measurement under several names: …" and placed below every real pair.

Read the verdict before the number. Two corrections, both of which raise the bar:

1. **A plant record holds far less evidence than its sample count.** Process signals hold their own value from one sample to the next, so a thousand samples of a slow signal can be worth a handful of independent observations. The evidence column shows that smaller number. Two signals that both drift upward over three days can correlate at `0.92` on the evidence of about three observations, and that demonstrates nothing.
2. **A large study makes many comparisons.** Among thousands of pairs, strong correlations appear by chance alone, so the floor rises with the number of pairs — and every lag you add is another comparison for every pair.

The consequence is that a *weaker* correlation can be the real finding: a valve that moves quickly carries real evidence at `+0.39`, while a slowly wandering line rate at `−0.28` does not.

A signal that never moved in the period is left out with a reason. A signal your account may not read narrows the study rather than refusing it, and the warnings say so.

> **Correlation is not cause.** Signals that share a shift pattern, an ambient temperature or an upstream setpoint correlate strongly with nothing between them. What this produces is a list of places to look, ordered so the redundant ones are not at the top. A lead is not a cause either: a tank level leads the pump that empties it.

Recording Timing Analysis

Recording ▸ Recording Timing Analysis… answers a different kind of question: were the timestamps in a recording any good? It works on a recording file (`.opslog` or `.opslog.gz`), not on the historian.

1. Press **Open recording…** and choose the file. If a recording is already open in playback, it is used.
2. In **Topic** choose a signal, or leave **auto (busiest signals)**.
3. Press **Analyse**. The whole recording is read, which can take a while.
4. **Save report…** writes the result as a Markdown file named after the recording, `…-timing-analysis.md` .

Against real hardware there is no ground truth for when a controller copied a value into its process image, so the analysis reconstructs the controller's scan grid from the value changes an over-sampled signal leaves behind, and checks that one address read through two acquisition groups produced intervals that agree. Both techniques can only *falsify*: the report prints the resolution of the reconstruction next to every coverage figure. The *Recording and playback* chapter describes recordings in full.

Worked example: investigating a slow night on the packaging line

The night shift on the packaging line reports that output was down, but no alarm stayed up long enough for anybody to note a time. This example follows the investigation path from a complaint to an armed rule. The times and values below are illustrative; yours depend on what the simulator did while the demonstration ran.

Before you start: open the packaging demonstration (**Help ▸ Demonstration…**, **Packaging line (simulated)**), start acquisition with **Acquisition ▸ Start**, and let it record for several hours with the historian enabled.

Step 1 — find the moment

1. Choose **Data ▸ Analysis Studio…**.

2. The **Something that happened** list already holds the events the detection rules raised in the last week, for example `2026-09-22 02:13:50 filler-fault (Packaging/Loader)` ("Pouch loader fault"). Since nobody knows which of them matters, sweep instead.
3. Set **Over the last** to `8h` and **…ending at** to `2026-09-22 06:00:00`, the end of the night shift. Leave **Same moment within** at `1m`.
4. Press **Find moments**. The status line reports, for example, "3 moments where several signals moved together, from 118 signals swept." The top of the list now reads `2026-09-22 02:14:05 · 5 signals moved together (7σ)`.
5. Read the footnote and the status line: any signal set aside as drifting or cycling is named there. A buffer level that fills and empties with every case is a typical one.

Step 2 — explain the moment

1. With the 02:14 moment selected, leave **Look back** and **Look forward** at `5m`, **Normal from** at `1h` and **Band (σ)** at `3`.

2. Press **Investigate**. The status line reads, for example, "5 of 118 signals were doing something different."

3. Read the **When** column, earliest first. An illustrative result:

Signal	Verdict	When
`Packaging/Loader/PouchInfeedRate`	Left its band	50 s before
`Packaging/Loader/CollatorBacklog`	Left its band	35 s before
`Packaging/Loader/Faulted`	Step change	5 s before
`Packaging/Loader/Rate`	Left its band	10 s after
`Packaging/Conveyors/BufferLoaderToCheckweigher`	Left its band	1.4 min after

   The infeed fell first, the collator backlog followed, and then the loader faulted. The rate and the buffer downstream moved *after* the occurrence: they are consequences. The candidate to look at on the floor is what feeds pouches to the loader.

4. Press **Make a report**. The status line says "Saved as 'investigation-20260922-0214'. Open it in the Report Studio." Attach it to the night shift's handover.

Step 3 — has the plant done this before?

1. Set **Over the last** to `7d` and clear **…ending at**.
2. Press **Find windows like this**. The five signals that moved form the shape, and the matches go to the top of **Something that happened**, for example `2026-09-20 23:41:30 · looks like this one (0.38)` and `2026-09-18 04:05:10 · looks like this one (0.44)`, followed by a gap to `1.9` and beyond.
3. The first two are much closer than the rest, so they are a family. Select each and press **Investigate** to confirm that the infeed led there too.

Step 4 — remember the fault

1. Select the 02:14 moment again and press **Investigate**, so the studio holds that investigation.
2. Press **Remember this fault…**. In **What the plant calls this** type `Loader starved of pouches` and press **Describe it**.

3. Read **The rule this produces**, for example `pouchInfeedRate <= 41.5 && collatorBacklog >= 12.8 && faulted >= 0.5`. Each threshold is the edge of the band the signal left; `Faulted` stepped, so it is cut at `0.5`.
4. Leave **Check the last** at `90d` and press **Has this happened before?**
5. Read the answer. If it reports the two nights found in step 3 and nothing else, the rule works and the fault is older than anybody knew. If the status line says the rule did not recognise its own occurrence, go back, investigate with **Look back** `2m`, and describe it again. If it reports twenty or more, the rule describes normal running: it was probably the `Rate` condition, and the rule needs a tighter threshold.
6. Press **Copy the rule**. Open **Acquisition ▸ Detection Rules…**, paste it, review the change, and activate the revision. From now on the plant raises an advisory when the loader is being starved, before it faults.

Step 5 — optional: what relates to what

To see whether the infeed and the backlog are related in general, not just on the night in question:

1. Press **What relates to what**. **Over** is filled in from the studio; set it to `7d`.
2. In **Lags, comma separated** type `30s, 2m`, and press **Correlate**.
3. Look for the pair `Packaging/Loader/PouchInfeedRate → Packaging/Loader/CollatorBacklog` with its lag and verdict. **Related** with a large evidence count supports the reading from step 2. **Not demonstrated** means a week of this line does not show it, however large `r` looks.

With the Web HMI module, the same investigation is available in the browser as **Analysis**: find the moment, rank what changed, describe and replay a fault rule, and study what relates to what, with the moved signals drawn around the investigated window.

When something is wrong

The occurrence list says it could not list recent occurrences. The node's detection rules or its event store could not be read. Type an instant, or use **Find moments**; neither needs a detected event.

"'8' is not a period this understands." Periods need a unit: `8h`, `24h`, `7d`.

Remember this fault… and Find windows like this stay greyed out. They need an investigation in which at least one signal moved. A window where nothing left its band has nothing to recognise it by.

Every signal is "No baseline". The historian holds nothing before the window, for example because recording started only minutes before. Choose a later occurrence, or a shorter **Normal from**.

The sweep finds nothing but the shift was clearly bad. The change was not a step. Read the signals named in the status line as set aside; a drift or an oscillation needs a trend or a correlation study, not a sweep.

A signal is missing from the results. The studio compares the signals the open project publishes, and your account must be allowed to read each of them. Open the right project, or ask an administrator about your account's signal scope.

Configuration and deployment

This chapter explains where a Meelmo installation keeps its configuration, how a change becomes the configuration the plant actually runs, how to go back to an earlier one, and how several machines are arranged around one plant. It covers the Desktop screens **Acquisition ▸ Node history…**, **Administration ▸ Deployment…** (with its certificate tool, the check that meets a re-created site authority, and the repair of a client that shows no history), **Administration ▸ Trusted clients…**, **Administration ▸ Fleet…**, **Administration ▸ Back up / restore node…** and **Recording ▸ Recorder Service…**, and ends with a short summary of what the installer sets up. The installation itself is described in the separate *Installation & Configuration Guide*.

The examples use the demonstration packaging line (`examples/demo/packaging.meelmoproj`), whose connection is called `Packaging` and whose signals include `Packaging/Checkweigher/CaseWeight`, `Packaging/Line/Rate` and `Packaging/Palletiser/Faulted`.

Projects and the node

Meelmo has two places a configuration can live, and it helps to keep them apart from the start.

	A project (`*.meelmoproj`)	**The node**
What it is	A file on disk: connections, acquisition groups and tags, virtual signals, alarm rules, recording preferences and the saved chart layout	The running authority for one plant: an Agent with its own database of accounts, configuration revisions and audit trail
How it changes	**File ▸ Save** / **Save As…** writes the file	A change is validated and *activated* as a new, immutable revision
History	None — the file is whatever was last saved	Every revision ever activated, with who activated it and when
Who may change it	Whoever can write the file	Only a signed-in person whose role grants the permission; every change is audited
Where it runs	Nowhere by itself — it is a document	On the machine that holds the node

Every Meelmo installation has a node. The Deployment screen says so in its own words: "an Agent that holds its accounts and configuration in a database it validates, versions and can roll back". The project file is how a configuration is carried, shared, e-mailed and archived; the node is where it runs.

What that means in daily use:

- **Opening a project on a workstation does not by itself change the plant.** On a Desktop attached to a node elsewhere, **File ▸ Open…** loads the file and then asks "*Make the node run this configuration? It replaces what the node is acquiring now, for every client attached to it, and is recorded as a revision you can roll back.*" (an Engineer or Administrator only). Decline, and the file is on this screen only; the status bar says "*Opened here only — the node is still running its own configuration. Apply it from I/O configuration to make the node run this one.*"
- **Applying is what sends it to the node.** Pressing **Apply** or **OK** in an editor such as **Acquisition ▸ I/O Configuration**, **Virtual Signals** or **Alarms** sends the whole configuration to the node, which validates and activates it. The status bar then reads *Configuration applied*.
- **Applying does not decide whether acquisition runs.** A node that was stopped stays stopped. A node that was running is running on the new revision afterwards (an I/O change stops and restarts acquisition around the activation; an alarm change leaves it running). An engineer who halted the line to change an address does not have it restart because they pressed Apply.

- **What a revision removes stops being current.** When a revision removes a connection, a tag or a computed signal, its last value is no longer presented as live anywhere: the node's current values (`realtime/current`, every HMI screen, an attached Desktop) show it as **Bad** with no value and the reason *Removed from the configuration in revision N*; the OPC UA server serves it as Bad and then drops it from its address space; MQTT, the node's own broker and Kafka are sent one last message with `"q":2`, `"v":null` and the reason in `"r"`; and Sparkplug sends the metric as null with a `Quality` property of 0. Signals the revision keeps are not touched — no gap, no change of quality. History is not rewritten: what was stored was true when it was stored.
- **What reads a removed signal is named.** A computed signal, alarm rule, recording trigger, detection rule or screen that reads a signal the revision does not have is listed as a warning in the validation panel and in the activation's result, with where it is. It is a warning, not a refusal — the rest of the revision works, and a reader left with nothing at all is reported **Bad** at run time with *Reads …, which the activated configuration does not have; there is nothing to compute it from* rather than sitting on its last value. The Desktop's I/O window asks about it before **OK** applies, and can point the readers at a tag of the same name, disable them or remove them (see *Chapter 2*).
- **Other screens follow.** When the node activates a revision — from this Desktop, from another workstation, from a browser or from a rollback — every attached Desktop reloads its editors from it and says "*The node activated a new configuration of '…'; the editors now show it*".

A project also does **not** carry where acquisition runs. That is a property of the machine (see *Deployment topologies* below), so the same `packaging.meelmoproj` opens standalone on a laptop and against a plant server without being edited.

Some things are the node's configuration and are never in the project file at all: the saved dataset and report definitions, the enabled modules and the list of trusted client certificates. The demonstration's *Also install its saved queries and reports* option exists for exactly that reason.

Configuration sections

The node's configuration is divided into **sections**. A section is the unit of schema version, validation and difference; it is **not** the unit of activation. Activation is always whole-node and atomic, so two sections can never disagree about which revision is live.

Section	What it holds
`node`	What the node is: its name, site and identity
`acquisition`	Connections, acquisition groups and tags — everything **I/O Configuration** edits
`analysis`	Signals the node computes from the ones it acquires (virtual signals)
`alarms`	Alarm rules and where their notifications are delivered
`automation`	Automation rules, the connectors they reach outside through, their helpers and the packages they came from
`recording`	What the node records, where, and on what condition
`semantic`	Detection rules: which executions, events and states the node recognises in what it stores
`hmi`	The published operator pages (drafts stay with the page editor)
`definitions`	Saved dataset queries and reports, kept together because a report names the datasets it renders
`historian`	Which historian this node writes to, and whether it writes at all
`modules`	Which licensed modules this site has turned on — enabled state only
`trust`	Which client certificates the node admits to its carrier, and which it has withdrawn

The order of the table is the order in which a difference between two revisions is listed: what the node is, what it acquires, what it derives, what it watches for, what it does about it, and what it keeps.

A section a build does not know is refused rather than ignored, so a newer project is never activated on an older node with part of it silently dropped.

Drafts, activation and revisions

Every change to a node goes through the same three steps. The Desktop performs them for you when you press **Apply**; the browser's Studios expose them more directly.

1. **Draft.** The node opens a draft based on the revision it is running. There is one draft per node. The draft remembers which revision it was based on.
2. **Save section.** Each edited section is written into the draft. Every save carries a version tag (an ETag). If somebody else has changed the draft since you read it, the save is **refused** with a "stale" message rather than merged — two people's configuration silently combined is a node running something neither of them wrote.
3. **Activate.** The node validates the whole draft. Findings are either **errors**, which block activation, or **warnings**, which are reported and do not. If nothing blocks it, the node writes an immutable **revision** and moves its active pointer to it in one transaction: either the node is running the new revision or it is running the old one, never something in between.

If activation is refused, the Desktop shows the node's own message in the status bar and the Event Log — for example which rule is broken, or that somebody else activated first — and puts its own copy of the configuration back the way it was, so the window does not go on showing a configuration the node rejected.

A revision is never modified afterwards. A **rollback** does not rewind anything: it writes a *new* revision whose content is copied from the old one. That is why a rollback can itself be rolled back, and why every state the node has ever run is still there to go back to.

Deleting what you made

Whatever a window or a page can create, the same window or page can delete: alarm rules, detection and advisory rules, virtual signals, connections, groups, tags and buffers, saved datasets, reports, fitted models kept in the project, glossary terms and owners, assets, asset types and signal roles, historical sources, saved searches, users, roles and service accounts on the Desktop; and users, HMI pages and projects, templates and composites, automation rules, reports, saved datasets, and kept models and rules in the browser.

One question, asked once. A deletion names what goes — one item by name, several as a list — and, from the node's lineage, **what reads it** and would read nothing afterwards: a computed signal that reads a deleted signal, an alarm rule on it, a flow, a screen, a report. The answers are:

Answer	What it does
Cancel	Nothing. It is the default button and the Escape key.
Disable instead	Offered for what has an on/off switch (alarm rules, virtual signals, assets, and automation rules from their menu). It stays, switched off.
Delete them too (N)	Offered when computed signals or alarm rules read what is going. They are deleted with it, in the same revision. Screens, flows and reports are only named: they are changed where they are edited.
Delete / **Delete anyway**	Deletes it. *Anyway* when something reads it: that reader stays and the node's validation warns about it at activation.

Several rows can be selected (Ctrl-click, Shift-click) and deleted with one question in the alarm, detection-rule, virtual-signal, saved-dataset and kept-model lists on the Desktop, and in the users and HMI page tables in the browser.

How it comes back. The question says. In a window with **OK** or **Save**, nothing is deleted until you apply the window, and closing it without applying keeps everything. Once applied, a deletion of anything in the node's configuration — rules, signals, I/O, datasets, reports, models, flows — is a new revision, and **rolling back** to the revision before it (below) brings it back. A user, a service account, a saved search and an HMI page are not revisioned: their question says *This cannot be undone*.

What is never deleted. The last enabled administrator; the account you are signed in with, from the browser; the node's own certificate and enrolment, from the node itself. Deleting a historical source or a historian setting removes the configuration only: the data it reads stays where it is. Deleting a dataset stops its scheduled drops; the files they wrote stay. Deleting an automation rule stops a run in progress when the deletion is activated, and its trace says it was cancelled. A page somebody else is editing is refused, with their name.

In the trail. Every activation now writes one `config.remove` entry per item it deleted — `alarms/rules 'Seal too hot' (r-17)` — with who activated it and which revision restores it; the data API records `api.dataset.delete`, `api.report.delete` and `api.project.*.stage`, and the web host `auth.user.deleted`. A refused deletion (yourself, the last administrator, a dataset a report reads) is recorded as refused.

From the API. `DELETE /api/v1/datasets/definitions/{name}` deletes a saved dataset; while a report reads it the answer is `409` naming the reports, unless `?force=true`. `POST /api/v1/project/remove` with `{ "kind": "model" | "rule" | "model-signal", "id": … }` writes the removal of a kept model, rule or model signal into the node's draft and returns the same review as keeping one; `POST /api/v1/project/drafts/{id}/activate` activates it. `DELETE /auth/admin/users/{username}` refuses (`409`) your own account and the last person who may administer.

Node history

Acquisition ▸ Node history… is the viewer for those revisions. It needs the Engineer role or higher.



Acquisition ▸ Node history on the packaging node: fourteen revisions, newest first, each with its number, time, account and comment (the demonstration datasets and reports saved, then its automation rules). Revision 13 is selected, so the right-hand side says one setting differs from the running revision 14 and shows the automation rules it would restore; Roll back to this is below

Part	What it shows
Revisions (left)	One row per revision, newest first: the revision number, the local date and time (`yyyy-MM-dd HH:mm`), the account that activated it, and **— running** on the one the node runs now. A second, smaller line shows the comment, or *Restored revision N* for a rollback
Summary line (top right)	*This is the configuration running now*, *Revision N is identical to the running revision M*, or *K settings differ between the running revision M and revision N*
Differences (right)	One row per changed setting, headed by its section and path, then *Added: …*, *Removed: …* or *old → new*
Roll back to this	Makes the selected revision the running one again (see below)
Close	Closes the window

On an installation whose configuration is only a project file, the window says "*This installation keeps its configuration in a project file, which has no history.*"

Selecting a revision always compares it with the **running** one, not with its predecessor, because that is the comparison that tells you what a rollback would change.

Worked example: reading a change and rolling it back

On the packaging line, an engineer widened the **Case weight out of tolerance** alarm, changing its condition from `abs([Virtual/Computed/WeightDeviation]) > 15` to `> 25` grams, and pressed **Apply**. The line now lets cases 20 g off target through without an alarm, and the shift lead asks for the old tolerance

back.

1. Open **Acquisition ▸ Node history…**. The top row is the revision just activated, marked **— running**, with the engineer's account beside it.
2. Select the row below it. The summary reads *1 setting differs between the running revision 14 and revision 13*, and the difference list shows the alarm rule's path in the `alarms` section with `> 25 → > 15`.
3. Check that this is the only difference. If other settings are listed, somebody changed more than the limit, and rolling back undoes all of it.
4. Press **Roll back to this** and confirm "*Roll back to revision 13? The node will restart acquisition on it. This is recorded as a new revision, so it can itself be rolled back.*"
5. The window reports *Rolled back to revision 13*. The list now has a revision 15 at the top, marked **— running**, with *Restored revision 13* under it. Every attached Desktop and browser screen picks up the old limit.

The rollback is audited under your name like any other activation.

Deployment topologies

One server owns the I/O, and every client is a view onto it. Two processes polling the same tag produce two different answers for the same instant, and neither of them is wrong. So a client never polls the controller, never recomputes the node's virtual signals and never runs its own alarm engine: it is sent what the node measured and what the node derived. History and alarms stay with the node that recorded them.

Mode	Means	Typical use
Standalone — this installation acquires	This Desktop holds its own node and acquires for itself	One engineer, one machine: commissioning, analysis, the demonstration
Server — acquires and serves clients	This Desktop acquires and lets other workstations attach	Trying out a shared setup; not for a plant (see below)
Client — a server acquires for it	Another machine holds the node; this Desktop is a view onto it	Engineers' and operators' workstations on a plant network

In a plant, the process that holds the node is normally the **recorder service** on a plant server, not a Desktop. It owns the drivers, runs with nobody signed in, survives a reboot, and serves every client from the same acquisition — the controller is polled once however many people are watching. The Deployment screen says this itself when you choose Server mode: "*Serving clients from this window is not what a plant should run: it stops when somebody closes the application. Configure the recorder service to serve instead.*"

The common shapes are:

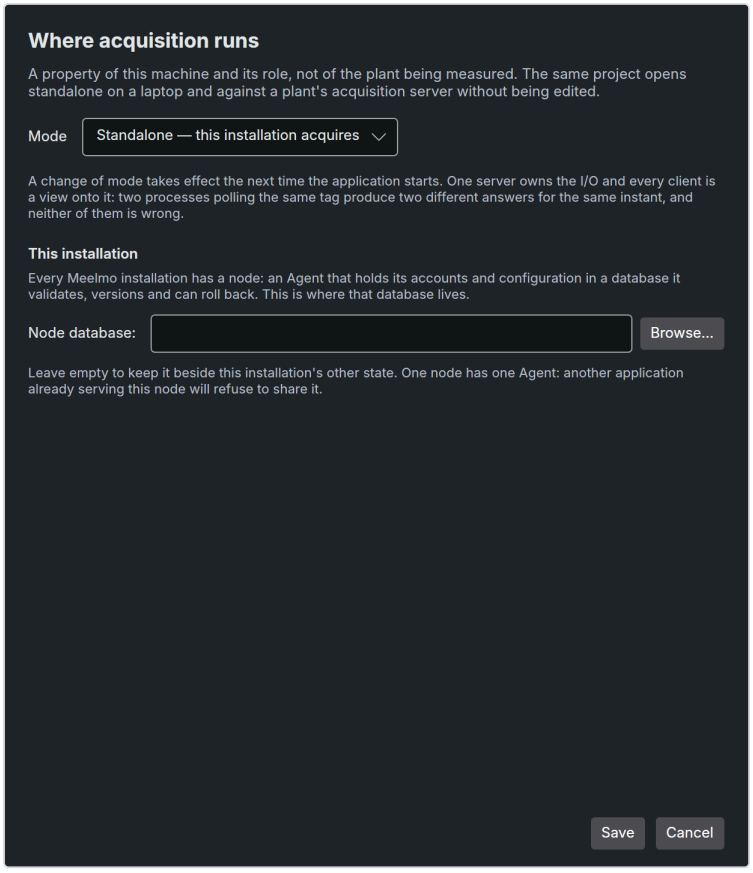
Shape	Who holds the node	Ports
Workstation (and browser) on one machine	The Desktop or the browser host	Loopback only; no TLS needed
Plant server on one machine	The recorder service	Loopback only; no TLS needed
Plant server with remote clients	The recorder service	`9701` (acquisition carrier) and `8444` (node control plane), both with TLS

The two ports on a server carry different things. The **carrier** port (`9701`) carries samples. The **node control port** carries sign-in, users, configuration and the audit trail; it is `8444` on a recorder service and `8443` where the node is served by the browser host. The single most common commissioning mistake is a Desktop pointed at `8443` when the node is held by a recorder, which serves `8444` .

With the Web HMI module, the browser host serves operator screens from the same node on port `8443` ; it is one more client of the node and does not poll the controller either.

Administration ▸ Deployment

Administration ▸ Deployment… sets this machine's role. It needs the Engineer role or higher. Its heading says what it is: *Where acquisition runs — a property of this machine and its role, not of the plant being measured.*



Administration ▸ Deployment in Standalone mode: the Mode list and, under This installation, where the node database lives.

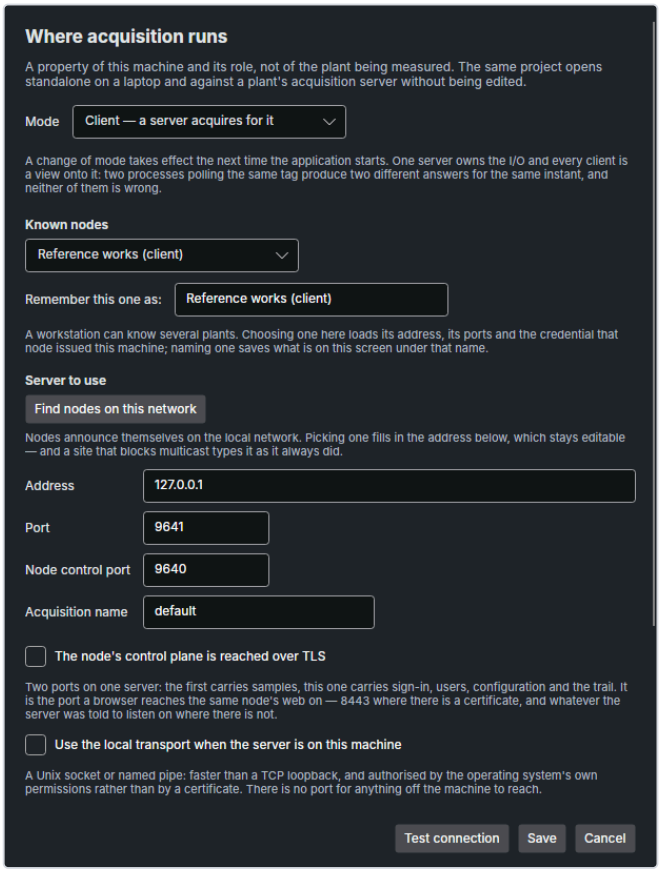
A change of mode takes effect the next time the application starts. Switching a running installation between hosting its own acquisition and being a client of somebody else's would tear down live acquisition, the historian and the recording underneath you, so **Save** says "*Saved. The new mode takes effect the next time this application starts*" and nothing changes until then.

Fields common to every mode:

Field	Meaning
Mode	*Standalone — this installation acquires*, *Client — a server acquires for it* or *Server — acquires and serves clients*
Node database: (under **This installation**)	The directory holding this installation's node database. Leave it empty to keep it beside the installation's other state (normally `C:\Meelmo\Data`). **Browse…** picks a folder. One node has one Agent: another application already serving the same node refuses to share it

Field	Meaning
Test connection	Signs in to the server with the details on screen and reports the result, without saving
Save / **Cancel**	Store the settings for the next start, or close without storing

In **Client** mode the screen asks which server to use:



Administration ▸ Deployment in Client mode: known nodes, network discovery, the server's address and ports, and the transport options.

Field	Meaning
Known nodes	Plants this workstation has been configured for. Choosing one loads its address, its ports and the credential that node issued this machine
Remember this one as:	Saves what is on the screen under a name, so a workstation can switch between several plants
Find nodes on this network	Nodes announce themselves on the local network. Picking one fills in the address below, which stays editable. On a network that blocks multicast, or from another VLAN, nothing answers — type the address instead
Address	The server's host name or IP address
Port	The acquisition carrier port, `9701` by default
Node control port	`8444` for a recorder service; `8443` for a node served by the browser host
Acquisition name	The name of the acquisition on the server; `default` unless the server runs several
The node's control plane is reached over TLS	Tick when the control port is served with a certificate — always the case for a server reached over the network

Field	Meaning
Use the local transport when the server is on this machine	Uses a named pipe or Unix socket instead of a TCP loopback: faster, and authorised by the operating system's own permissions. Nothing off the machine can reach it
Sign-in on the server: User / Password	An account on the server. The server decides this client's role from its own user store. The password is sealed with this machine's key, so a copied settings file carries a credential that will not open elsewhere; when one is stored the field reads *stored — leave blank to keep it*
TLS: Pinned server certificate	For a self-signed server on a plant network with no PKI: pin that exact certificate. There is deliberately no "accept any certificate" setting
Client certificate / **Client certificate password**	The certificate this workstation presents, where the server requires one

In **Server** mode it shows the listener instead:

Field	Meaning
Listen on / **Listen port**	Where clients connect. Loopback by default: a server that is reachable from the plant network the moment it is installed is not a decision anyone made
Require TLS for clients that are not on this machine	Leave ticked. Anything listening on a routable address without a certificate refuses to start
Server certificate (PKCS#12) / **Certificate password**	The certificate the server presents; the password is *sealed on this machine*
Certificate…	Opens the certificate tool (see *The certificate tool* below): inspect the file, **Create self-signed…**, **Import .pfx…** or **Import PEM…**. On a plant server whose site runs its own certificate authority, the tool says the certificate *renews itself* and nothing needs doing
Require clients to present a certificate as well	Mutual TLS: only workstations with an admitted certificate may attach (see *Trusted clients*)

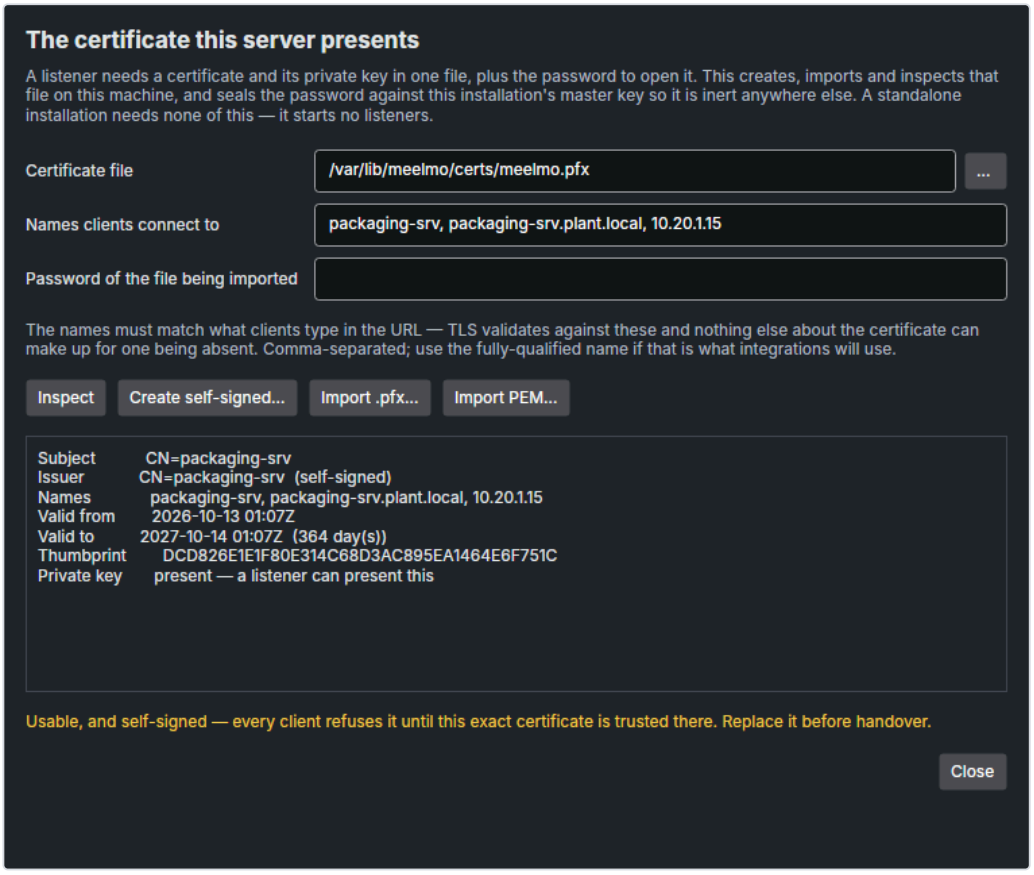
The client's own credential. When a client deployment is saved and the workstation holds no data credential yet, it asks the node for one of its own ("*Asking the node for this workstation's own credential…*"). Nobody types a token. If a workstation shows live values but no history, or no alarms, that credential is usually what is missing; see *When a client shows no history* below.

Seats. Each attached client takes one licensed client seat, which the status bar shows — for example *Attached to plant-srv-01 as j.meyer (Engineer) · seat 1 of 2*. A standalone installation takes no seat. If every seat is in use, **Test connection** reports it and the client is refused. A seat is a 30-second lease that the client renews while connected, so a workstation that crashes gives its seat back on its own. The number of seats is part of the licence (see *Licensing and modules*).

The certificate tool

Certificate… in Server mode opens **TLS certificate**, headed *The certificate this server presents*. A listener needs a certificate and its private key in one file, plus the password that opens it; this window creates, imports and inspects that file on this machine. A standalone installation never needs it, because it starts no listeners.

Where the site issues its own certificates, a block at the top says so first: *This site issues its own certificates*, the site's name, when the machine's certificate renews itself, and the names it is accepted for. If the last renewal failed, the reason is shown in red and the certificate already in use goes on being served. Issuing, enrolling, withdrawing and replacing an authority are done in the browser's **Administration ▸ Certificates**; the rest of this window is for supplying a certificate of your own instead.



The TLS certificate window after Create self-signed... for packaging-srv, packaging-srv.plant.local and 10.20.1.15: the certificate file, the names, the import password, the Inspect, Create self-signed..., Import .pfx... and Import PEM... buttons, the details read from the file, and the amber verdict Usable, and self-signed

Field or button	What it does
Certificate file	Where the PKCS#12 file is kept. It starts at `meelmo.pfx` in the `certs` folder of this installation's state, or at the file Deployment already names; **...** picks another place. When the file exists, the window inspects it as it opens
Names clients connect to	The host names and addresses clients will type, comma-separated. It starts with this machine's name and its best local address, because on a plant network clients are often pointed at the IP. TLS validates against these names and nothing else
Password of the file being imported	Only for **Import .pfx...**: the password the file came with
Inspect	Reads the file and lists **Subject**, **Issuer** (marked *self-signed* where it is), **Names**, **Valid from**, **Valid to** (with days left), **Thumbprint** and **Private key** (present or absent), then gives a verdict
Create self-signed...	Makes a new certificate for the names above, after asking you to confirm, and warns if it would replace the file that is there
Import .pfx...	Copies a `.pfx` or `.p12` from your own PKI into the certificate file, opening it with the password above

Field or button	What it does
Import PEM…	Asks for the certificate (`.pem` , `.crt` or `.cer`), then its private key (`.pem` or `.key`), and combines them into the certificate file. This is the case a real certificate authority usually leaves you in, and it saves you needing `openssl` on a server
Close	Returns to Deployment

The verdict under the details is one of:

Verdict	Means
Usable: it has a private key, a name and time left.	Good to serve
Usable, and self-signed — every client refuses it until this exact certificate is trusted there. Replace it before handover.	Shown in amber: fine for a pilot
Expired on … / Not valid yet. Check this machine's clock.	Every client refuses it
No private key in this file, so a listener cannot present it.	Import the file that includes the key
No names in this certificate, so every client refuses it whatever else is right.	Create or obtain one with names

Password sealing. Whichever way the file was made, the tool makes the password itself, seals it against this installation's master key and hands both the file path and the sealed password back to Deployment: **Server certificate (PKCS#12)** is filled in and **Certificate password** reads *sealed on this machine*. You never see or type that password, and a copy of the settings on another machine cannot open it. Press **Save** in Deployment to keep it.

A self-signed certificate encrypts the connection and authenticates nothing: the confirmation says so, and every client refuses it until that exact certificate is pinned there (**TLS: Pinned server certificate**). Use it for a pilot, not for handover.

When the site authority has changed

A workstation pins the site's root certificate the first time it joins a node, and from then on refuses a node that presents a different one. **Test connection** and **Save** both check this before they send anything. When the root differs, the **Site authority changed** dialog opens, headed *This node presents a different site authority than the one this workstation trusts*.

This is what you see when a plant server was reinstalled, its node rebuilt, or the workstation moved to another plant. It is also exactly what somebody on the network pretending to be the server would show you, and the next thing the workstation would send is a password. So the dialog makes you compare before anything changes.

Part	What it shows
Trusted by this workstation	The root this workstation pinned: its subject, **SHA-256** fingerprint and **SHA-1 thumbprint**
Presented by *host*	The root the node presents now, in the same form
SHA-256 fingerprint of the new root	Where you type or paste the fingerprint you read on the plant server itself

Part	What it shows
Replace	Disabled until what you typed is the presented root's whole SHA-256 fingerprint. Colons, spaces and capitals do not matter; a prefix, the SHA-1 thumbprint or the old root's fingerprint does not count, and the dialog says *That is not the SHA-256 fingerprint of the root this node presents*
Cancel	Keeps the old root. `Esc` cancels; `Enter` presses nothing, so the dialog cannot be dismissed by a reflex

To confirm a genuine change:

1. On the plant server itself, open **Administration ▸ Certificates** in the browser and read the root's *Thumbprint*: it is the SHA-1 shown in the dialog. The server's trust page, `https://localhost:8443/trust` opened on that machine, shows the SHA-256.
2. If they match, type or paste the SHA-256 into the dialog and press **Replace**.
3. The workstation asks the node again and pins the new root only if the node still presents it. Then the test or save carries on.

A fingerprint read only from the dialog proves nothing, because the dialog is what an impostor would show too. If you cancel, Deployment says *Nothing was replaced. This workstation still trusts the authority it had, and asked the node for nothing.* If the node presented no root the workstation could read, the dialog has no **Replace** button and says so; an administrator can then install the site's new root in the machine's trusted roots, or delete the pinned file, and try again. Nothing on the workstation re-pins by itself.

When a client shows no history

A client workstation reads the node's history, and watches its alarms, with a credential of its own: a read-only machine account the node creates for it, like a service account, named *Workstation (machine name)* and limited to `ReadCatalog`, `ReadHistory`, `ReadRealtime` and `ReadEvents`. Live values do not need it, so a workstation without one looks healthy until you ask it for history. Then the status bar reads, in red:

Historian: unavailable — This workstation has no credential for the node's data API, so it can show no history. Open Deployment and save the connection again to ask the node for one.

To repair it:

1. Open **Administration ▸ Deployment…**. It still shows the client settings.
2. Under **Sign-in on the server**, check **User** and enter the **Password**. The workstation asks the node with this account, so the account must exist on the node and be allowed to create machine accounts.
3. Press **Save**. The line under the buttons reads *Asking the node for this workstation's own credential…*.
4. If the node issues one, the credential is stored, sealed with this machine's key, the window closes and the Desktop reconnects to the server with it. As the mode has not changed, no restart is needed.

If the node refuses, the Deployment window says why and saves the rest anyway, so the workstation goes on showing live values:

Message	What to do
This workstation has no password for the node, so it cannot ask for a credential of its own…	Enter the password under **Sign-in on the server** and save again
The node refused the credentials for '…'…	The user or password is wrong on the node; correct them

Message	What to do
This node does not administer machine principals...	The node cannot issue one; history is not available to this workstation
This workstation could not obtain a credential from the node: ...	The reason follows, for example the node could not be reached; fix it and save again

The credential is asked for only while the workstation holds none. The same missing credential is behind *This workstation is not watching the node's alarms: it has no data credential for that node...* in the status bar and the Event Log, and the same repair applies.

Worked example: attaching a workstation to the packaging line server

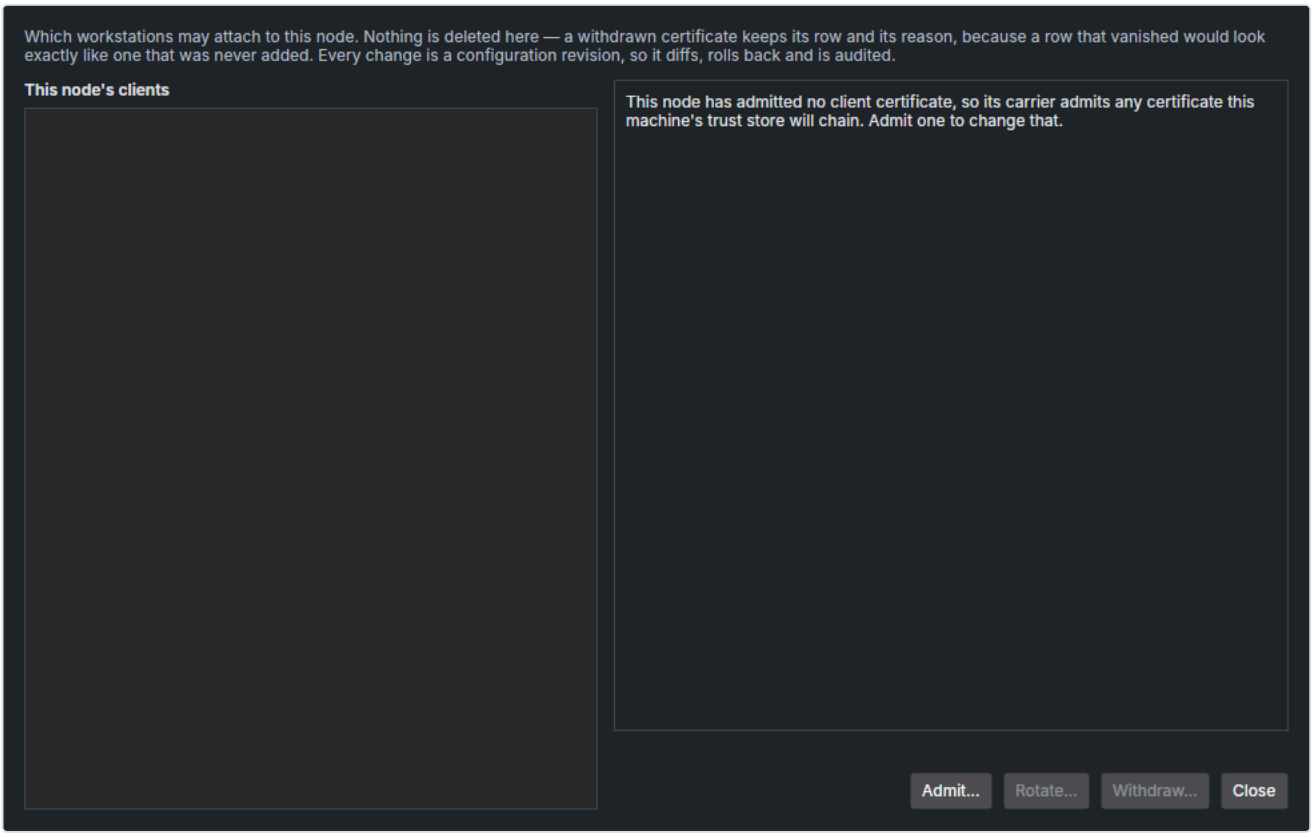
The packaging line runs on a plant server `pkg-srv-01` under the recorder service. An engineer's laptop is to become a client.

1. On the laptop, open **Administration ▸ Deployment...** and set **Mode** to *Client — a server acquires for it*.
2. Press **Find nodes on this network**. If the server answers, pick it; otherwise type `pkg-srv-01` into **Address**.
3. Leave **Port** at `9701`, set **Node control port** to `8444` (a recorder service), and tick **The node's control plane is reached over TLS**.
4. Enter an account on the server under **Sign-in on the server**, for example `j.meyer`.
5. Type `Packaging line` into **Remember this one as:** so it can be picked from **Known nodes** later.
6. Press **Test connection**. A good answer reads like *Connected to pkg-srv-01 as j.meyer (Engineer) · acquisition 'default' · seat 1 of 2*.
7. Press **Save** and restart the Desktop. The status bar now names the server and the seat, the signal tree shows `Packaging/…`, and **Start** and **Stop** act on the server's acquisition.

If the test says the remote certificate was rejected, the laptop does not trust the site's certificate authority yet; on machines installed by Meelmo Setup this is done for you, otherwise pin the server's certificate under **TLS**.

Trusted clients

Administration ▸ Trusted clients... lists which workstation certificates the node admits to its acquisition carrier. It needs the Administrator role.



Administration ▸ Trusted clients on a node that has admitted no client certificate yet, with the Admit, Rotate and Withdraw buttons.

While the list is empty the node's carrier "*admits any certificate this machine's trust store will chain*". Admitting the first certificate changes that: from then on only admitted certificates get through.

Nothing is ever deleted from this list. A withdrawn certificate keeps its row and its reason, because a row that vanished would look exactly like one that was never added. Every change is a configuration revision in the `trust` section, so it appears in **Node history**, can be rolled back and is audited.

Part	Meaning
This node's clients (left)	One row per certificate, marked **on**, **expired** or **withdrawn**
Details (right)	**Thumbprint**, **Admitted**, **Added**, **Note**, **Expires** (with days left), **Replaced**, **Withdrawn** and **Reason**, and a sentence saying whether a handshake with it gets through
Admit…	Choose the workstation's certificate file, give it a note you would recognise on a list (for example `Line 3 operator desk`), check the thumbprint and confirm
Rotate…	Replace the selected certificate with a new one. The old one is withdrawn in the same revision, so the two cannot come apart
Withdraw…	Refuse the selected certificate from the moment the change is activated. A reason is required — "*Without one, nobody can tell a stolen laptop from a typo.*"
Close	Closes the window

An expired certificate is refused at the handshake by TLS itself; rotate it.

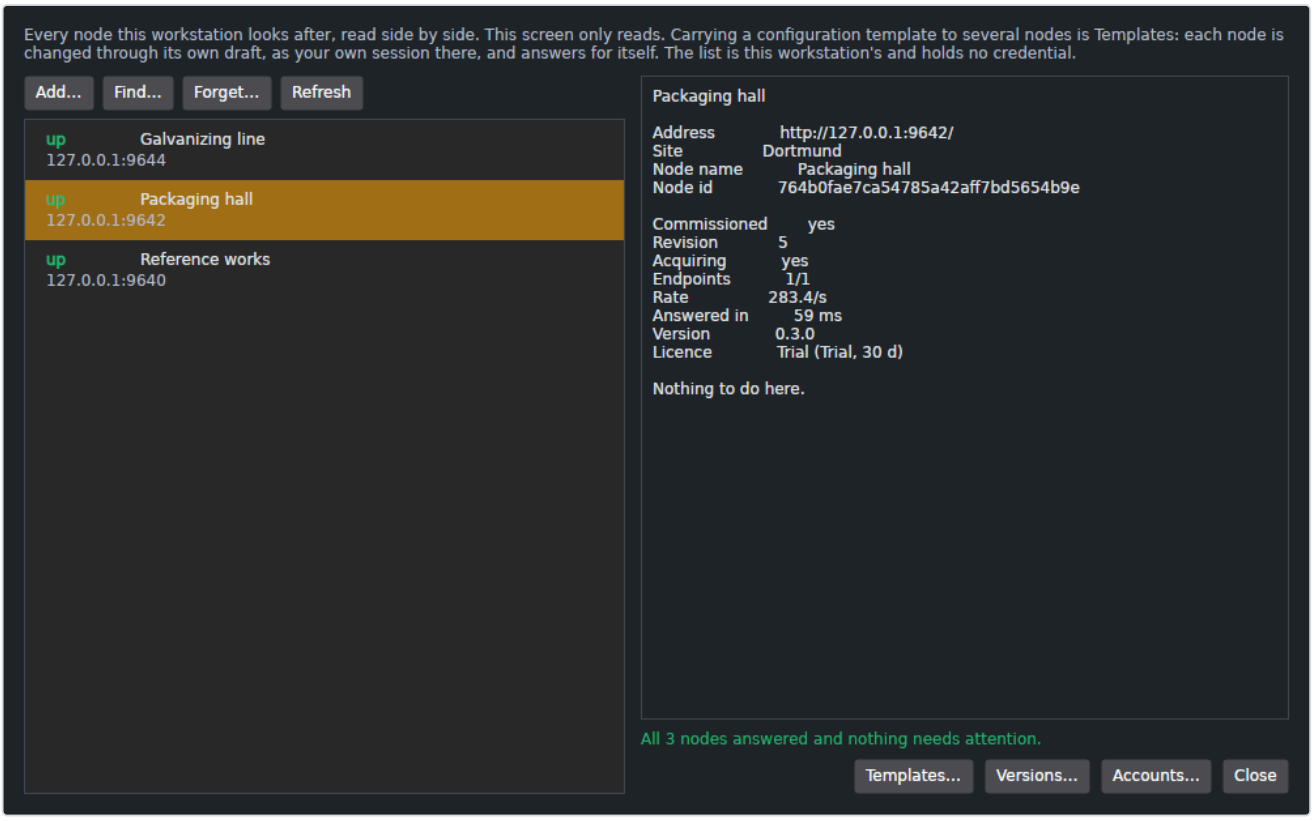
Worked example: a stolen laptop

The laptop `pkg-eng-02` was stolen from the packaging line office.

1. Open **Administration ▸ Trusted clients…** on the node (or on a Desktop attached to it).
2. Select the row noted `pkg-eng-02` and check the thumbprint.
3. Press **Withdraw…**, enter the reason `Laptop reported stolen, incident 2026-114`, and confirm.
4. The row stays, marked **withdrawn**, with the reason and the time. The node refuses that certificate at its next handshake and says so to whoever presents it.

Fleet

Administration ▸ Fleet… shows every node this workstation looks after, side by side. It needs the Engineer role or higher.



Administration ▸ Fleet with three plants on the roster — the galvanizing line, the packaging hall and the reference works — all up; the packaging hall selected, with its site, node id, revision, acquisition, rate, version and trial licence; and the Templates, Versions and Accounts buttons

The Fleet window itself **only reads**. Six nodes are six authorities: carrying a configuration to several of them is the **Templates…** screen (below), where each node is changed through its own draft, validation and activation, as your own session on that node, and answers for itself. The list belongs to the workstation and holds no credential.

Control	What it does
Add…	Asks for a name for the list and an address (`host` , `host:port` or a URL)
Find…	Listens for nodes advertising themselves on the network and adds them
Forget…	Removes the selected node from this workstation's list. The node itself is untouched
Refresh	Asks every node again
Attach…	Makes this Desktop a client of the selected node (you sign in to it separately)
Accounts…	Opens *Accounts across the fleet* (below)

Control	What it does
Templates…	Opens *Configuration templates* (below). Not in the Community Edition
Versions…	Opens *Versions across the fleet* (below)
Node list (left)	One row per node, marked **up**, **check** or **down**
Details (right)	**Address**, **Site**, **Answered**, **Reason**, **Last seen**, **Node name**, **Node id**, **Commissioned**, **Revision**, **Acquiring**, **Endpoints**, **Rate**, **Answered in**, **Version** (and whether it is behind the newest in the fleet) and, once you are signed in to that node, **Licence**
Status line	*All 3 nodes answered and nothing needs attention*, or *1 of 3 nodes needs attention*

If an address answers as a different node from last time, the details warn that somebody may have re-used the address.

Accounts… is a report of who exists on which node, and where the nodes disagree — for example a maintenance account that is an Engineer on two plants and disabled on a third. **Only where they disagree** filters the list. It never creates, changes or removes an account anywhere: that is done on the node that owns it.

Where a node runs on a failover cluster, add the cluster's address, not a machine's. A row is keyed by host and port, so an entry naming whichever machine is active today reports a healthy plant as down after a failover.

Configuration templates

Fleet ▸ Templates… keeps configuration templates and carries them to the roster's nodes. Fleet management is part of the **Professional** edition: a template is deployed only to a node whose own licence is Professional or a running trial, and a Community build offers no Templates button.

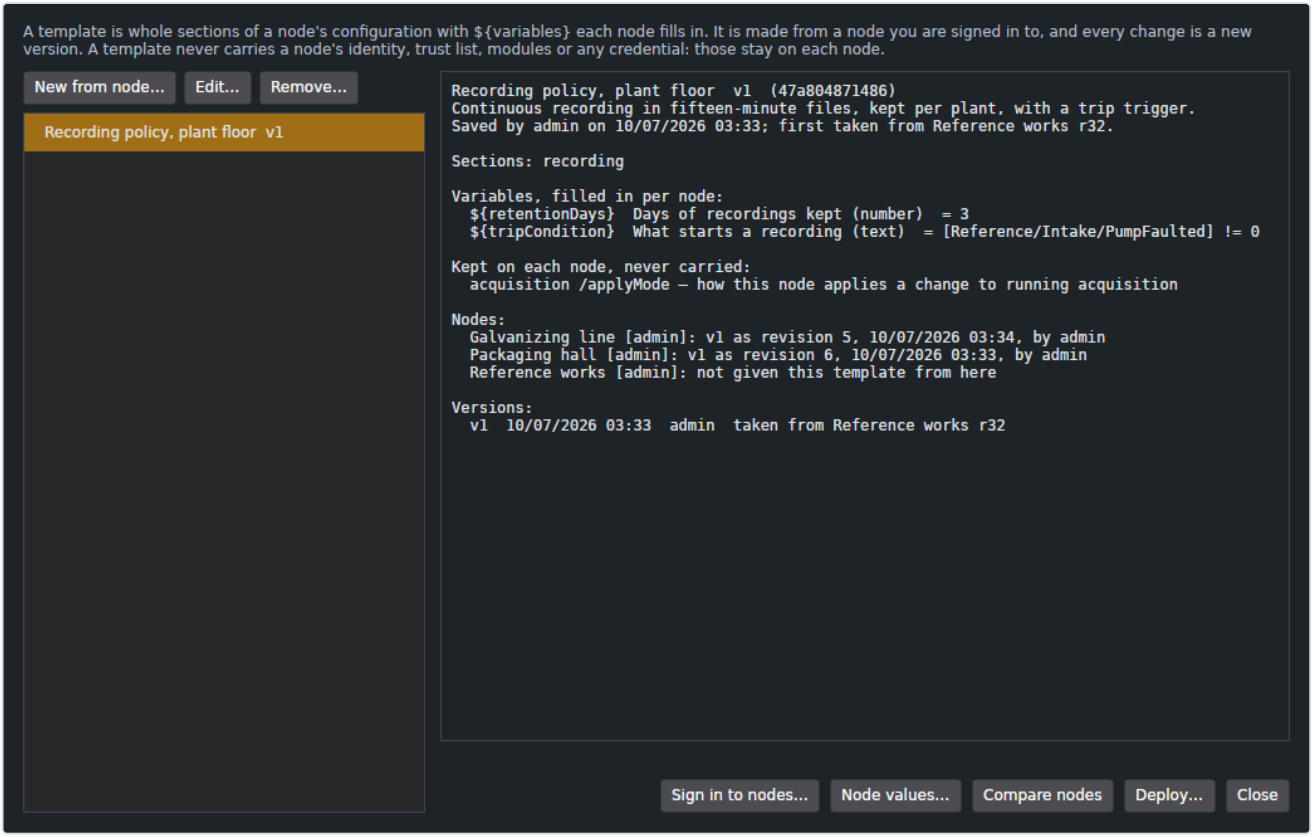
A **template** is whole sections of a node's configuration — acquisition (connections, groups, tags), analysis, alarms, automation, recording, semantic rules, operator screens, saved queries and reports, historian — written in the node's own JSON, with **variables** that each node fills in. It never carries a node's identity (`node`), its trust list (`trust`) or its modules (`modules`), and never a credential.

Control	What it does
Sign in to nodes…	Sends one account and password to each chosen node. Each node decides on its own and issues its own session; the password is not kept, and the sessions last until the application closes or the node ends them
New from node…	Makes a template from a node you are signed in to: choose the node and the sections. Every sealed secret in them becomes a *kept value* (below)
Edit…	Edits the sections' JSON, the variables and the kept values, and saves a **new version** — the old ones are kept. **Make a setting a variable…** turns one setting (a JSON Pointer such as `/endpoints/0/settings/IpAddress`) into a variable whose default is its current value
Node values…	Each node's values for the variables: its PLC address, its line name
Compare nodes	Reads each node that was given the template, and each you are signed in to, and says whether it still runs exactly what the template gives it — or which settings were edited at the node since (**drifted**)
Deploy…	Opens *Deploy* (below)

Variables. Written `${name}` inside a JSON string. Declared one per line as `name | label | text, number or boolean | default`; no default makes the value required on every node. A string that is exactly one number or boolean variable becomes a JSON number or switch. `${node.name}`, `${node.site}` and `${node.host}` come from the roster without being declared, and `$${` writes a literal `${`. A value is placed inside the string it was written in, so a quote in a line name cannot add a setting. A variable whose name says *password*, *secret* or *token* is refused.

Kept values. Settings each node keeps whatever the template says, copied from that node's own active configuration at deploy time: every sealed credential (which only the node that sealed it can open), and how the node applies a change to running acquisition — so a deployment uses **partial apply** unless that site chose full. A node that has no value at a kept setting is refused with *Set it on the node first*.

The details show, for every node, which version of the template this workstation gave it, as which revision, and whether that is an older version.



Fleet ▸ Templates with Recording policy, plant floor: taken from the reference works' recording section, with ${retentionDays} and ${tripCondition} filled in per node, the acquisition apply mode kept on each node, and the two plants it was deployed to with the revision each runs now

Deploy and staged rollout

Templates ▸ Deploy… carries one template version to the nodes you tick.

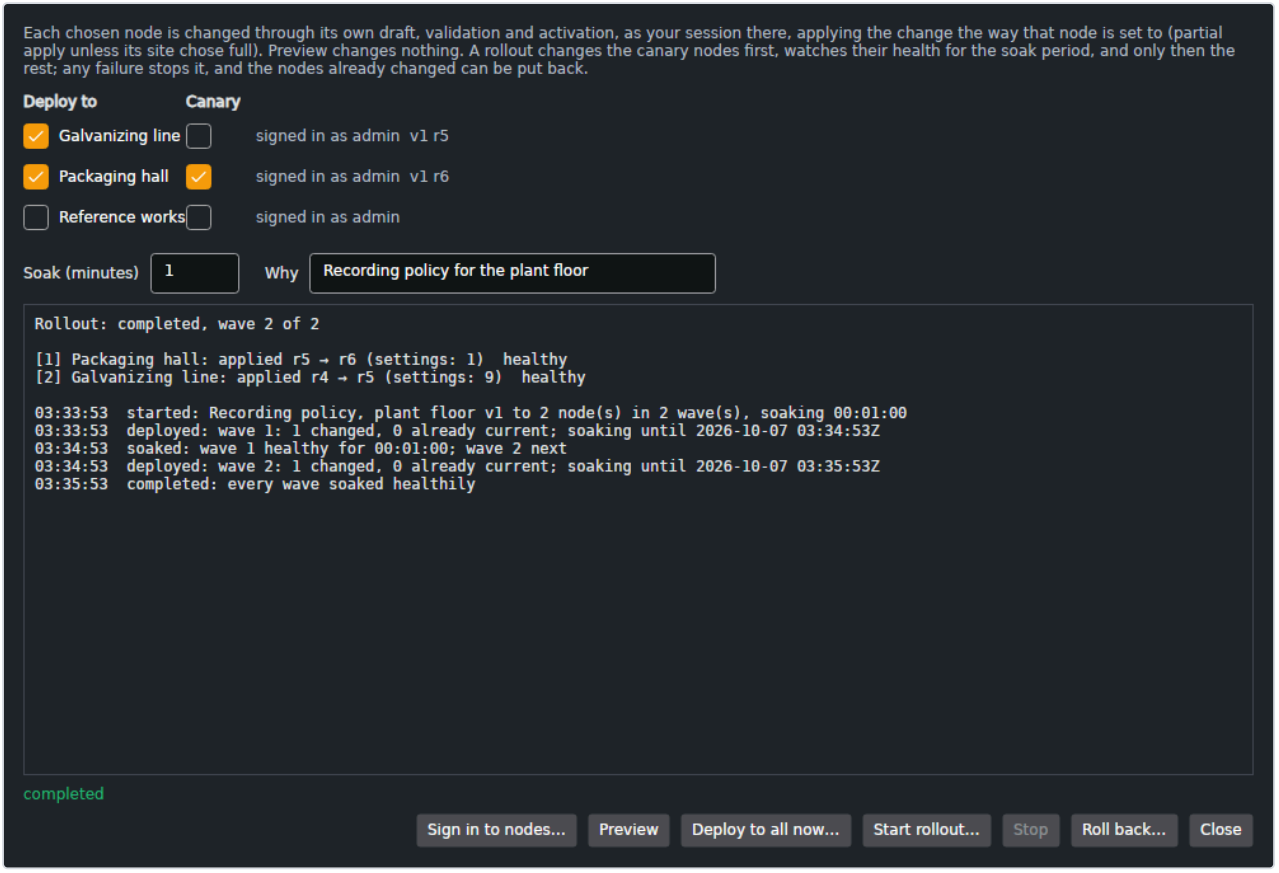
Control	What it does
Preview	Asks each node what it would change and whether it would accept it. Nothing is activated
Deploy to all now…	Each node, at most four at a time: its draft is written, validated and activated, as your session there
Start rollout…	A staged rollout: the nodes ticked **Canary** first; then, after their health has stayed good for **Soak (minutes)**, the rest

Control	What it does
Stop	Stops a rollout. Nodes being changed at that moment finish first; the rest are not touched
Roll back…	Puts back, on each node changed here, the revision it ran before

Each node answers for itself, on its own line: *applied* (with the revision it ran and the one it runs now), *already current*, *refused* (in the node's words — its validation, a missing value, an edition that does not include fleet management), *not permitted*, *not signed in*, *did not answer*, *answers as another node*, *did not come up*. A node that does not permit you to change configuration is refused **before anything is written**: you need **Config.Read**, **Config.Io** and **Config.Activate** on that node, and for a rollout also **Diagnostics.Read**, because its soak reads the node's health. A node whose draft already holds somebody else's change in progress is refused and that draft is left alone.

The soak judges each changed node against how it was doing just before the change: it fails when the node stops running the deployed revision (it rolled itself back, or somebody changed it), when acquisition that was running stops, or — after the first minute — when an endpoint that was connected is not. **Any failure halts the rollout** before the next node is touched; the nodes already changed are left as they are until you choose **Roll back…**.

A rollback is the node's own: a new revision carrying the old configuration. A node that was changed again after the deployment is refused, by name — roll it back from its own history (*Revision history*) after looking at what changed.



Templates ▸ Deploy after a staged rollout: the packaging hall as the canary and the galvanizing line after it, a one-minute soak, each node applied with its old and new revision and healthy, and the timeline from start to "every wave soaked healthily"

Closing this window, or the workstation, during a rollout leaves it halted where it was, with its rollback offered: a workstation keeps its node sessions only while it runs. A rollout that must outlive the person who started it belongs on the browser host (*Administration ▸ Fleet* in the web administration

chapter), which keeps its people's node sessions across a restart and carries an interrupted rollout on.

Each node's revision comment names the template, its version and content hash. This workstation's audit trail records every template saved, sign-in, deployment, rollout step and rollback under your name; each node records its own side in its own trail.

Versions across the fleet

Fleet ▸ Versions… lists which product version and edition each node runs, and flags the nodes behind the newest version present in the fleet. **Save upgrade files…** writes, for each node behind, the upgrade instructions and an answer file for the existing unattended installer (`--unattended --answers` for Meelmo Setup, `/ANSWERS=` for the Windows installer), with **Role**, **NodeName**, **Site**, **Administrator** — and an **empty** `Password=` **line** for the operator to type at the machine.

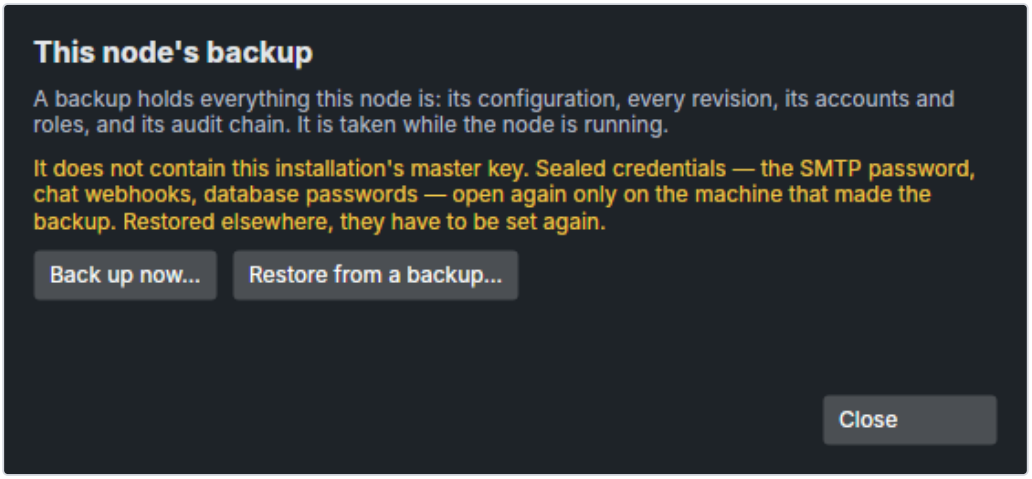
Upgrading remains an operator's action. Nothing is downloaded, copied or run on a node from here: there is deliberately no self-updater, because a program that fetches and runs binaries on every plant at once is the largest attack surface a fleet feature could add.

Fleet alerts

Fleet alerts are sent by the **browser host**, which runs unattended — see *Administration ▸ Fleet* in the web administration chapter. The Desktop shows the same facts in the Fleet window while it is open.

Backup and restore

Administration ▸ Back up / restore node… copies the node, and puts one back. It needs the Administrator role.



Administration ▸ Back up / restore node…: This node's backup, what a backup holds, the amber note that it does not contain the master key, and the Back up now… and Restore from a backup… buttons

Control	What it does
Back up now…	Asks where to write the file and writes it while the node keeps running. The result reads *Written to … (N KB), from node '…'*
Restore from a backup…	Asks for a backup file, shows which node it came from and when, and stages it

What a backup contains: everything the node is — its configuration, every revision, its accounts and roles, and its audit chain.

What it does not contain:

- **The master key.** Sealed credentials — the SMTP password, chat webhooks, database passwords — open again only on the machine that made the backup. Restored on a different machine they must be entered again, and the restore says so before you confirm.
- **The sample data.** History belongs to the historian and is backed up with that database or store.
- **The licence files**, which are bound to the machine and are kept in the data folder.

A restore takes effect at the next start. It replaces the node's configuration, revisions, accounts and audit trail, and is staged until the application (or the service holding the node) restarts, because the node has its database open.

For a complete recovery of a plant server, back up the data folder (normally `C:\Meelmo\Data`) **together with the master key inside it**, plus the historian.

The public address

Some of what Meelmo sends out links back to the browser client: a scheduled process digest (see *Reports and analysis in the browser*, with the Web HMI module) links each finding to the process study that shows it. Inside the browser a path such as `/analytics/process?…` is enough. In a PDF on somebody's desk, or a workbook attached to a mail, it opens nothing — the file has no address it came from. The **public address** is the address people's browsers reach the web host at, and links that leave the browser are written against it.

Host	In its configuration file	Environment (wins over the file)
Web host	`network.publicAddress` in `web-config.json`, also set on **Administration ▸ Branding ▸ Public address**	`MEELMO_WEB_PUBLIC_ADDRESS`
Recorder service (when it holds the node)	—	`MEELMO_WEB_PUBLIC_ADDRESS`

From the browser. An administrator sets the web host's address on **Administration ▸ Branding**, in the **Public address** panel, without a shell on the server. The panel shows the address saved in `web-config.json`; when `MEELMO_WEB_PUBLIC_ADDRESS` is set on the host, a warning above the box gives its value and says it wins over the saved one. Under the box it says what the host is using now — the address it started with, or none — and, when the saved address differs from that, that it is used from the host's next start: the host reads the address when it starts, and the node's report schedule is handed it then. **Save address** checks the address by the same rule the host reads the file with (below) and says what is wrong in the reader's language; a saved address is written into `web-config.json` with the rest of the file left as it was, and recorded in the audit trail as `web.publicAddress.saved` with the address before and after. An empty box clears it. Both reading and saving need the Administer permission.

It is also a name on the certificate. Where the site issues its own certificates, the web host's certificate carries the public address's host beside the machine's own names and addresses, and is re-issued at the next start when it does not. From outside a plant a browser dials that address — a name in the plant's DNS, a router's address, a VPN name — and a certificate that leaves it out fails there even on a machine that trusts the site's root: the browser warns, the page can be clicked through, and the live values, which a browser does not let past a certificate it was clicked through, never arrive. With a certificate file of your own (**Certificate…** in the Desktop, or one from the plant's authority) the host cannot add the name, and says at every start when the certificate does not cover the public address.

Give the whole address, with a path if a reverse proxy serves the client under one: `https://meelmo.plant.example` or `https://plant.example/meelmo/`. It must be `http` or `https` and carry no query or fragment. It is empty by default, and Meelmo does not guess it: a host bound to `0.0.0.0`, behind a proxy, or known by a name in the plant's DNS does not know the address its readers use.

Scheduled reports are produced by whichever process holds the node, so set it where the node is held: in `web-config.json` when the web host holds it, and as `MEELMO_WEB_PUBLIC_ADDRESS` in the recorder service's environment on a plant server where the recorder holds it. A report rendered by hand from the browser as a PDF or a workbook uses the web host's. An address that cannot be read is logged as a warning at start and not used; the host still starts.

Without it, links stay paths. The reports still render, and a scheduled or downloaded PDF or workbook with a digest says under its table that its links do not open from the file.

The recorder service

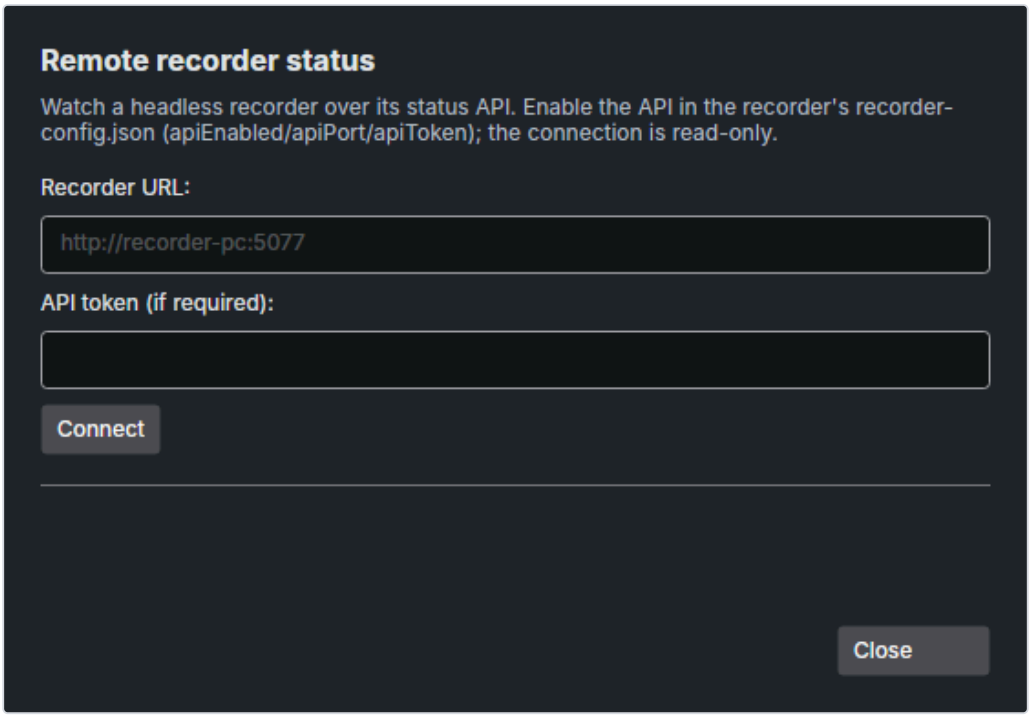
The recorder service is the headless process that holds a plant's node: it acquires, records and serves clients with nobody signed in, and restarts with the machine. It is included in the Professional edition (and during the trial).

Recording ▸ Recorder Service… generates the service's configuration from the open project and installs, starts and stops the Windows service. It needs the Engineer role or higher. Its configuration belongs to the machine — which port, which certificate on this box — and lives in `recorder-config.json` in the data folder. The window has five tabs:

Tab	What it sets
Recording	**Output folder**, **File prefix**, **Rotate at (MB)**, **Rotate at (minutes)**, **Keep for (days)**, **Compress rotated files (.gz)**, **Lossless (never drop samples)**, and topics to include or exclude
Status API	A read-only HTTP status interface (`GET /health` , `GET /status`): **Enable the status API**, **Bind host**, **Port** and **Bearer token**. Loopback by default; a routable bind requires a token
Data access	**Serve the data-access API** from this process, its host, port, gRPC, certificate and trusted reverse proxies. Serving the MCP surface also needs the industrial-integration module
Acquisition	**Serve acquisition to desktop clients**: **Listen on**, **Port**, the certificate, **Also accept clients on this machine over the local transport** and **Require clients to present a certificate as well**. Attaching clients sign in with the node's own accounts
Service	The **Config file** path, **Save config**, the service **Status:** and **Install**, **Uninstall**, **Start**, **Stop** and **Refresh**

Service management from this window is available on Windows; on Linux the service is a systemd unit set up by the installer.

Recording ▸ Recorder Status… watches a recorder over its status API from any workstation: enter the **Recorder URL** (for example `http://pkg-srv-01:5077`) and, if required, the **API token**, and press **Connect**. It shows the recorder's state, version, uptime, current file, samples recorded and dropped, trigger and alarms. The connection is read-only.



Recording ▸ Recorder Status, where a recorder's status API address and token are entered.

During a trial, the recorder service runs unattended for 8 hours at a time and then stops acquisition until it is restarted. A licensed installation runs continuously.

What the installer sets up

The installer asks what the machine is for (**Workstation**, **Plant server**, **Client of a Meelmo server elsewhere** or **Decide later**) and where the plant's data lives, and configures the machine from the answers; nothing has to be registered by hand afterwards. Running a newer installer over an installed one upgrades in place and leaves the data folder untouched.

The *Installation, setup and the programs* chapter covers this in full: the Meelmo Setup wizard page by page, repair and removal, servers with no screen, answer files, Linux, containers, the Community Edge VM, and the programs' command-line options. See *Installation, setup and the programs*.

Users, roles and security

This chapter covers who may use a Meelmo installation and what each person may do: user accounts and roles, signing in and out, changing your own password, the tamper-evident audit trail, rotating the encryption key that protects stored credentials, the certificates that secure connections between machines, and the service accounts that integrations use. It describes the Desktop screens under the **Administration** menu. With the Web HMI module, the same administration is also available in the browser.

The examples use the demonstration packaging line and its node, whose seeded accounts are `admin` (Administrator) and `shift` (Operator).

How identity works in Meelmo

Accounts belong to the **node** — the authority that holds a plant's configuration (see the *Configuration and deployment* chapter). Every surface signs in against the same accounts: the Desktop on the node's own machine, a Desktop attached as a client, the browser and the service accounts used by integrations. One account, one role, one audit trail, whichever screen was used.

- The **first administrator** is created when the node is commissioned: on the first-run screen of a workstation, or by the installer on a plant server. There is no other way to create that first account.
- **Every action that changes configuration or commands the process is authorised** by the node against the signed-in account, not by the screen. A button the Desktop enables is a convenience; the node still decides.
- **Every such action is written to the audit trail.** If the entry cannot be written, the action is blocked rather than performed unrecorded — the Desktop says "*The action was blocked because it could not be recorded in the audit log*".

Signing in

When sign-in is required, the Desktop opens with the sign-in window.



The Desktop's sign-in window, with the Username and Password fields and the Sign in and Exit buttons.

Field or button	Meaning
Username:	The account name, for example `shift`

Field or button	Meaning
Password:	The account's password
Sign in	Signs in. On success the status bar shows *Signed in as shift (Operator)*
Exit	Closes the application without signing in

The sign-in protects itself:

- **Password policy.** At least 8 characters and 3 of: lowercase, uppercase, digit, symbol. A password that does not meet it is refused wherever it is set.
- **Failure throttling.** Repeated failures are limited per minute. Once the limit is reached the window says "*Too many failed attempts. Try again in N s.*"
- **Idle sign-out.** A session that has been idle for 30 minutes is signed out, and the sign-in window comes back. A click or a key press counts as activity. The 30 minutes is the default: each site sets its own at the node, from 1 to 1,440 minutes, or 0 for never — on the Desktop under **Administration ▸ Users & Security…** (see *Idle sign-out* below), in the browser under **Administration ▸ Users & permissions**. The node applies it to every surface signed in to it, and a change reaches sessions that are already open.
- **Password storage.** Passwords are never stored. The node keeps a salted PBKDF2-SHA256 hash (210,000 iterations).

Users and roles

Administration ▸ Users & Security… manages accounts. It needs the Administrator role.



Administration ▸ Users & Security on a node: the Require sign-in option, the account list and the account buttons, including Assign roles, Manage roles and Permissions.

Control	What it does
Require sign-in (enable user accounts & audit trail)	Turns accounts and the audit trail on. On a node this is always on and cannot be switched off — the node's refusal is shown if you try
Account list	One row per account: *name* · *role*, for example `admin · Administrator`
Add user	Asks for a username and a password. **New accounts start as Viewer** (least privilege); raise the role explicitly
Remove	Deletes the selected account
Reset password	Sets a new password for the selected account (subject to the policy)
Change role	Moves the selected account one step round the four built-in roles: Viewer → Operator → Engineer → Administrator → Viewer
Assign roles…	On a node: tick the exact set of roles the account holds, including roles you have defined yourself
Manage roles…	On a node: define your own roles beside the four built-in ones (see below)
Permissions…	On a node: what this person may do beyond their roles (**Extra permissions:**), and which part of the plant they are scoped to (**Area**; blank means the whole plant)
Enable / disable	Disables the account without deleting it, or enables it again. A disabled account cannot sign in, and its history in the audit trail is kept
Verify audit log	Checks the audit chain (see *The audit trail*)
View audit log	Opens the audit log viewer
Close	Closes the window

Every change made here is itself audited.

An older installation whose accounts are kept in a file rather than in a node shows only the first seven buttons; it has the four built-in roles and no others.

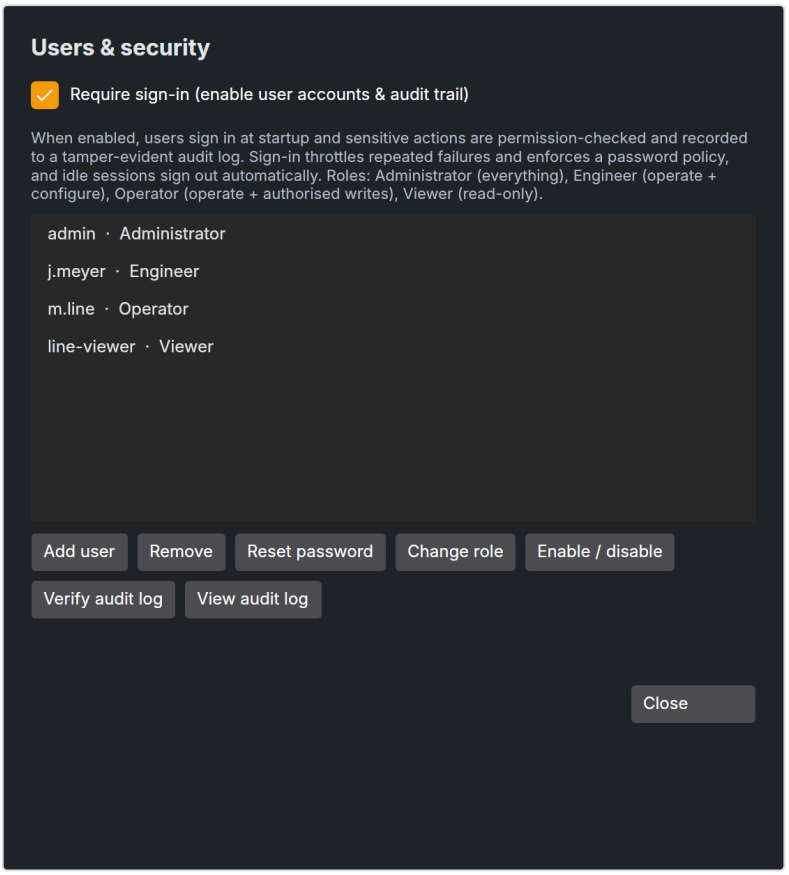
Idle sign-out

On a node, the window also shows **Idle sign-out, in minutes:**, a number box and **Apply**. It is the node's setting, so it applies to everybody signed in to the node — at this Desktop, at a Desktop joined to the node, and in a browser.

Value	Meaning
1 to **1440**	Signed out after this many minutes without a click or a key press. A larger number is held at 1440 (a day)
30	The default, and the security baseline
0	Never. The window warns that this departs from the security baseline, which signs idle sessions out after 30 minutes

The box is filled from the node when the window opens. **Apply** stores the value at the node; only somebody who may administer accounts (the Administrator role) may change it, and the node refuses anybody else, whatever the screen shows. The new value applies at once, including to sessions that are already open: a Desktop locks at its next check and the node refuses a session that has been idle for

longer than the new value. Each change is in the node's audit trail as `security.session_idle`, with the old and the new value and who made it. A Desktop joined to a node too old to have the setting keeps 30 minutes.



An account list with one account in each built-in role: admin as Administrator, j.meyer as Engineer, m.line as Operator and line-viewer as Viewer.

The four built-in roles

The four built-in roles form a ladder. Each one includes everything the role below it may do.

Role	In one sentence
Viewer	Read-only: open projects, trend, play back recordings, browse history and export
Operator	Viewer, plus run the system: start and stop acquisition and recording, and — where explicitly authorised — write to the process
Engineer	Operator, plus configure: I/O, virtual signals, alarms, recording, the historian, detection rules, deployment and the recorder service
Administrator	Everything, including accounts, service accounts, licensing, modules, trusted clients, backup and restore, and key rotation

What each role may do on the Desktop

The Desktop checks the following before it opens a screen or performs an action. The node checks again when the action reaches it.

Action (Desktop menu)	Viewer	Operator	Engineer	Administrator
Open a project, trend, play back, export (**File**, **View**)	yes	yes	yes	yes

Action (Desktop menu)	Viewer	Operator	Engineer	Administrator
Acquisition ▸ Start / **Stop**	—	yes	yes	yes
Recording ▸ Start Recording / **Stop Recording**, recording settings and trigger	—	yes	yes	yes
Authorised writes to a PLC tag	—	yes	yes	yes
File ▸ Save / **Save As…**	—	—	yes	yes
Acquisition ▸ I/O Configuration, applying a project to the node	—	—	yes	yes
Acquisition ▸ Virtual Signals	—	—	yes	yes
Acquisition ▸ Alarms	—	—	yes	yes
Acquisition ▸ Node history… (and rollback)	—	—	yes	yes
Acquisition ▸ Detection Rules, **Data ▸ Historian**, semantic model import and export	—	—	yes	yes
Administration ▸ Deployment…, **Fleet…**	—	—	yes	yes
Recording ▸ Recorder Service…	—	—	yes	yes
Administration ▸ Users & Security…, **Service Accounts…**	—	—	—	yes
Administration ▸ Modules…, **Trusted clients…**	—	—	—	yes
Administration ▸ Back up / restore node…, **Commissioning report…**	—	—	—	yes
Administration ▸ Rotate encryption key…	—	—	—	yes
Help ▸ License ▸ Install License…	—	—	—	yes

When a role does not allow an action, the Desktop shows *Not permitted* with "*Your role (Operator) is not permitted to: …*", and the refusal is audited.

The node's permissions

On a node, a role is a named set of permissions. The four built-in roles come with the following sets, which cannot be edited or removed.

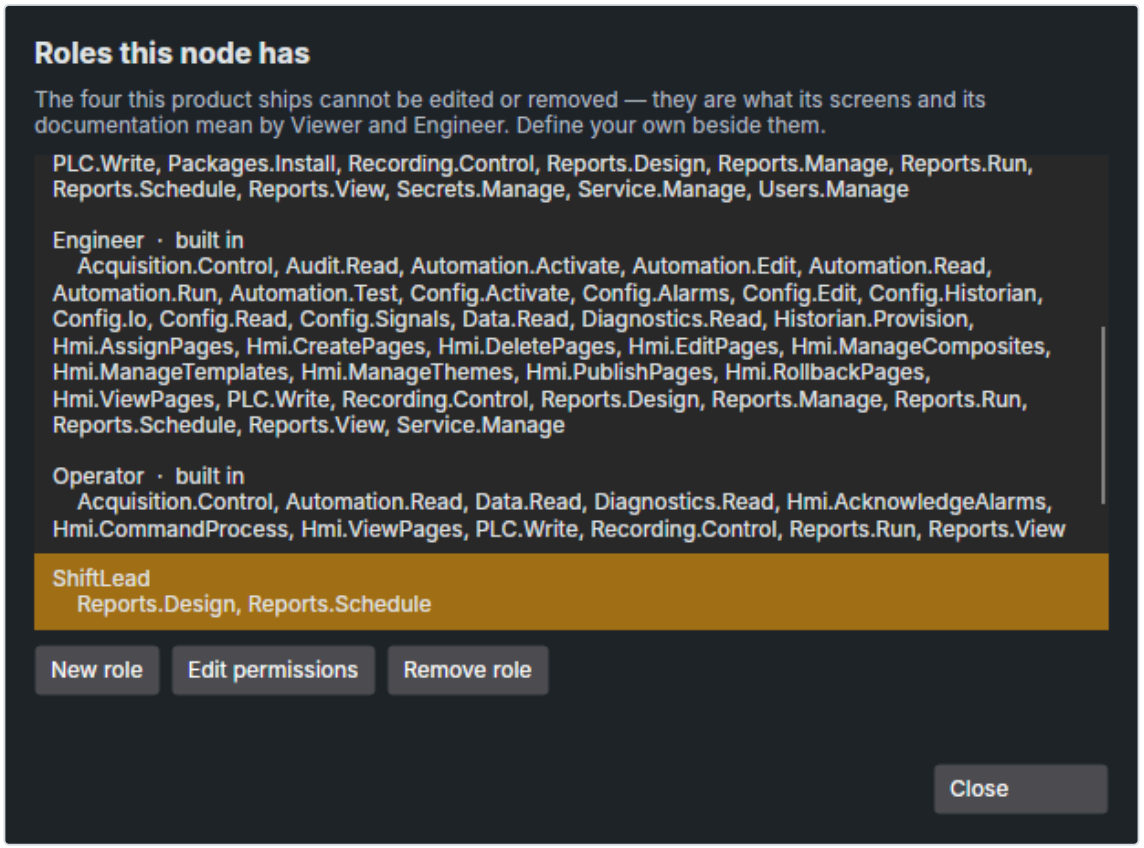
Permission group	Viewer	Operator	Engineer	Administrator
`Data.Read`, `Diagnostics.Read`	yes	yes	yes	yes
`Acquisition.Control`, `Recording.Control`, `PLC.Write`	—	yes	yes	yes
`Config.Io`, `Config.Alarms`, `Config.Signals`, `Config.Historian`, `Config.Edit`, `Config.Read`, `Config.Activate`, `Historian.Provision`, `Service.Manage`, `Audit.Read`	—	—	yes	yes
`Users.Manage`, `License.Install`, `License.Manage`, `Secrets.Manage`, `Node.Backup`, `Commission`, `Control`, `Node.Claim`	—	—	—	yes
Reports: `Reports.View`	yes	yes	yes	yes
Reports: `Reports.Run`	—	yes	yes	yes

Permission group	Viewer	Operator	Engineer	Administrator
Reports: `Reports.Design`, `Reports.Manage`, `Reports.Schedule`	—	—	yes	yes
Automation: `Automation.Read`	—	yes	yes	yes
Automation: `Automation.Edit`, `Automation.Activate`, `Automation.Run`, `Automation.Test`	—	—	yes	yes
Automation: `Connectors.Configure`, `Packages.Install`	—	—	—	yes
HMI pages: `Hmi.ViewPages`	yes	yes	yes	yes
HMI operation: `Hmi.CommandProcess`, `Hmi.AcknowledgeAlarms`	—	yes	—	yes
HMI authoring: `Hmi.CreatePages`, `Hmi.EditPages`, `Hmi.DeletePages`, `Hmi.PublishPages`, `Hmi.RollbackPages`, `Hmi.ManageTemplates`, `Hmi.ManageComposites`, `Hmi.ManageThemes`, `Hmi.AssignPages`	—	—	yes	yes
`Hmi.BreakEditLease`	—	—	—	yes

The HMI and automation permissions matter only where the Web HMI module is enabled. Note that the built-in Engineer role authors operator pages but does not command the process or acknowledge alarms from them; give an engineer who also operates the line the Operator role as well, with **Assign roles…**.

Defining your own roles

Manage roles… opens *Roles this node has*. The four built-in roles are listed and marked **built in**; define your own beside them.



Roles this node has on the packaging node: the built-in roles with their permissions and, selected, the ShiftLead role of the worked example below with Reports.Design and Reports.Schedule; New role, Edit permissions and Remove role underneath

Button	What it does
New role	Asks for a **Role name** and creates an empty role
Edit permissions	Opens *Permissions for …*: tick the permissions the role carries
Remove role	Removes a role you defined (after confirmation; it cannot be undone)

Where custom roles and extra permissions take effect. They are held and enforced by the node, so they apply wherever the node checks a permission itself: the browser screens (with the Web HMI module), reports, automation and every change the node is asked to make. The Desktop's own menus are gated on the account's built-in role — the one shown in the account list — so a person who must use a Desktop screen such as **Acquisition ▸ Alarms** needs a built-in role that includes it.

Worked example: a shift lead on the packaging line

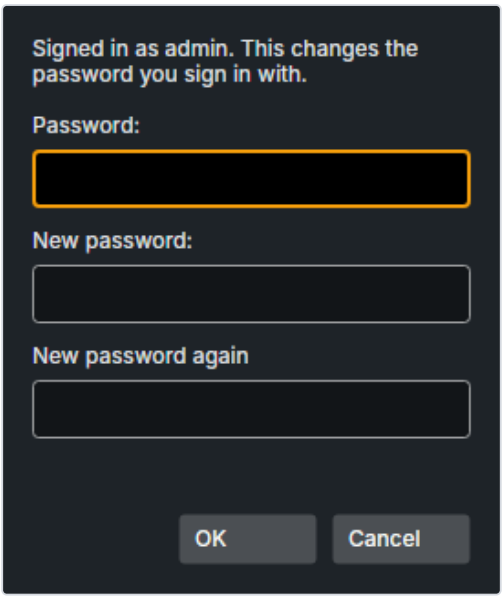
The packaging line's shift lead, `k.brandt`, runs the line from the Desktop, and in the browser must also run, design and schedule the shift report — but must not configure I/O or manage accounts.

1. In **Administration ▸ Users & Security…**, press **Add user**, enter `k.brandt` and a password that meets the policy, for example `Line4-Shift!`. The account appears as `k.brandt · Viewer`.
2. Press **Change role** once. The account becomes `k.brandt · Operator`, which lets it start and stop acquisition and recording on the Desktop.
3. Press **Manage roles…**, then **New role**, and name it `ShiftLead`.
4. Select `ShiftLead`, press **Edit permissions**, and tick `Reports.Design` and `Reports.Schedule`. Close the window.
5. Select `k.brandt`, press **Assign roles…**, and tick both `Operator` and `ShiftLead`.

6. Optionally press **Permissions…** and fill in **Area** to scope the account to one part of the plant; leave it blank for the whole plant.
7. Close the window. The node's audit trail now records the account's creation (`users.create`), the new role (`roles.create`, `roles.set_permissions`) and the assignment (`users.set_roles`), each under the administrator's name.

Changing your own password

Administration ▸ Change my password… changes the password of the account you are signed in with. Any role may use it.



Administration ▸ Change my password: the current password, the new password and the new password again.

Field	Meaning
Password:	Your current password
New password:	The new password; it must meet the policy
New password again	The same new password, to catch a typing mistake
OK / **Cancel**	Change it, or leave it unchanged

The window names the account at the top (*Signed in as …*). When the change succeeds, **every session the account had open is closed**, on every screen, so sign in again with the new password. If the current password is wrong the window says so; after repeated attempts it asks you to wait.

On an older installation whose accounts are kept in a file, only an administrator can change passwords, with **Reset password** in **Users & Security**.

Signing out, sessions and seats

Administration ▸ Sign out ends your session and returns to the sign-in window without closing the application. Use it when leaving a shared workstation. Someone else can then sign in on the same Desktop; the Desktop re-attaches to the node as that person. Pressing **Exit** on the sign-in window closes the application.

A session also ends by itself after 30 minutes without activity, and when its password is changed.

On a Desktop attached to a server as a client, signing out also **releases the client seat** it held, so another workstation can use it. Seats are the licence's count of concurrent clients; the status bar shows the one held (for example *seat 1 of 2*). A seat is also given back on its own within about 30 seconds if a workstation disappears without signing out.

The audit trail

Meelmo keeps two tamper-evident trails. Both are **SHA-256 hash chains**: every entry carries the hash of the one before it, so an entry that is edited, removed or inserted breaks every hash after it.

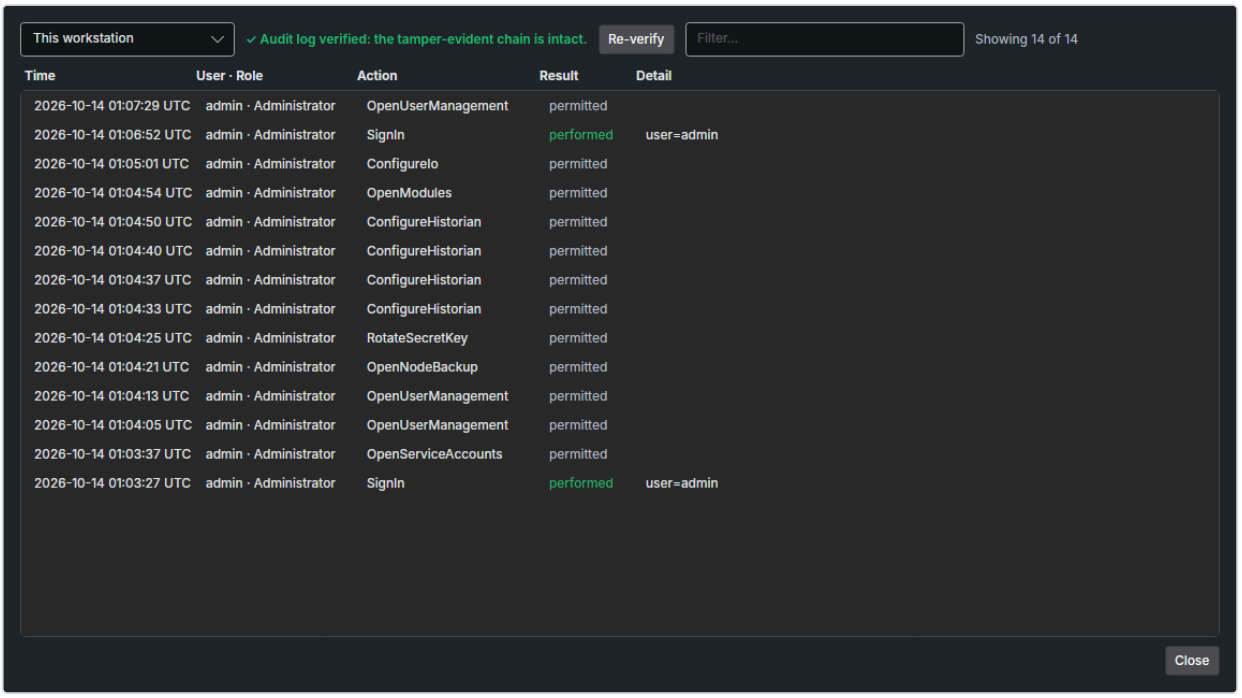
Trail	Kept by	Records
The node's trail	The node, in its database	Everything that reached the node from any screen, under the person who did it: revisions activated and rolled back, users, roles and permissions, service accounts, alarms acknowledged and shelved, backups, trusted clients, licences, modules, and refusals
This workstation's log	Each installation, as daily files under its `audit` folder	What happened at this machine: sign-ins to it, sign-outs, process writes sent from it and its own administration, such as key rotation

PLC writes are audited even when sign-in is switched off.

Verifying and viewing

In **Administration ▸ Users & Security…**:

- **Verify audit log** checks the chain and reports either "*Audit log verified: the tamper-evident chain is intact.*" or "*Audit log verification FAILED — the chain is broken in: …*" with the file and line where it breaks and why (an entry changed after it was written; an entry removed, reordered or inserted; or a writer that began a new chain, which older versions did after a power loss and is usually a fault rather than an edit).
- **View audit log** opens the viewer. A selector chooses between **This workstation** and **Node '…'**. The columns are **Time**, **User · Role**, **Action**, **Result** and **Detail**; **Filter…** narrows the list, **Re-verify** checks the chain again, and the count beside them reads *Showing N of M*.



View audit log on a workstation joined to the packaging node, This workstation chosen in the selector: the chain verified, Re-verify, the Filter box and Showing 14 of 14, then the sign-ins and the windows the administrator opened, each permitted or performed

For the node, the node verifies its own chain and reports, for example, "*Node 'packaging' verified its own chain: 1 284 entries intact.*" The chain detects an entry that was edited or removed without being recomputed; it is not proof against somebody who can rewrite the database, which is why the node's machine must be protected like any other server.

Reading the node's trail needs the `Audit.Read` permission (Engineer and Administrator). A person without it is told the chain was *not checked*, which is not the same as intact.

With the Web HMI module, **Administration ▸ Audit** in the browser shows both trails with filters.

Worked example: who changed the reject threshold?

The reject gate on the packaging line started rejecting good cases overnight.

1. Sign in as `admin` and open **Administration ▸ Users & Security… ▸ View audit log**.
2. Choose **Node 'packaging'** in the selector.
3. Use **Filter…** with `config.activate` to list configuration activations (a rollback is recorded as `config.rollback`).
4. The entry at `02:14` shows `j.meyer · Engineer`, action `config.activate`, result `ok`, with the revision in **Detail**.
5. Open **Acquisition ▸ Node history…**, select the revision before it and read the difference; roll back if needed (see the *Configuration and deployment* chapter).

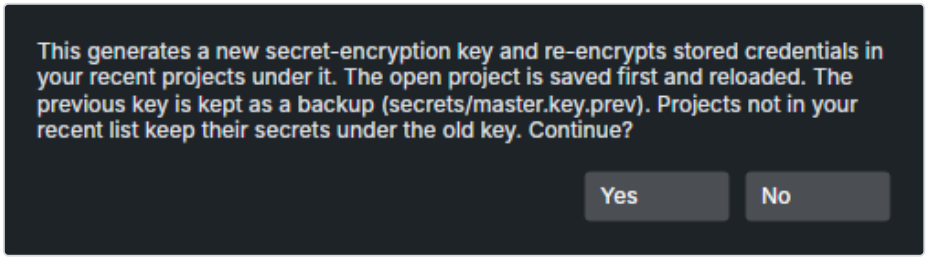
Stored credentials and key rotation

Passwords the product must use — driver passwords, the SMTP password, notification webhooks, database passwords, certificate passwords, a client's server password — are **sealed** with AES-256-GCM under a master key bound to this machine. A configuration file never holds a password, only a sealed token. On Windows the master key is protected by the operating system at machine scope, so the Desktop and the recorder service share it; on Linux and macOS the key file and the secrets folder are owner-only.

A consequence to remember: **a configuration copied to another machine cannot unseal its secrets there.** That is intended. Drivers connect without the credential, mail is sent without authentication, and so on, until the secrets are entered again on the new machine. A node backup restored elsewhere behaves the same way.

Administration ▸ Rotate encryption key… (Administrator) generates a new master key and re-seals stored credentials under it:

1. Take a backup of the data folder first.
2. Choose **Administration ▸ Rotate encryption key…** and read the confirmation: the open project is saved first and reloaded, credentials in your recent projects are re-encrypted, and the previous key is kept as `secrets/master.key.prev`.



The confirmation shown by Administration ▸ Rotate encryption key…, with Yes and No

3. Confirm. The result reads "*Rotation complete: re-encrypted N secrets across M project files.*"

Rotation is all-or-nothing: if any secret cannot be read with the current key, it stops with "*Key rotation was aborted and nothing was changed*". Projects that are not in the recent list keep their secrets under the old key, which `master.key.prev` can still open. The rotation is audited.

Never e-mail the master key or put it in a support bundle; support bundles and logs exclude it, and credential-like text is masked before it reaches a log file.

Certificates and TLS

Anything listening on a network address other than loopback requires TLS (version 1.2 or 1.3); a component without a certificate refuses to start on such an address. There is no "turn encryption off", and Meelmo's own channels have no "accept any certificate". A few integrations with other software do, off by default and logged as a warning while on, and joining a node trusts the site's authority on the first contact only; the *Security & Compliance Report*, §6, lists each.

The site's own certificate authority. A plant server creates a root and an issuing certificate authority on first start and issues itself a certificate valid for 90 days, renewed automatically 30 days before expiry with nobody signing in. Every listener presents the renewed certificate from its next connection on; nothing needs restarting. Meelmo's own programs on machines installed by Meelmo Setup trust that authority automatically. With the Web HMI module, **Administration ▸ Certificates** in the browser manages the authority, enrols machines and withdraws certificates, and serves a trust page for browsers.

The certificate a Desktop server presents. In **Administration ▸ Deployment…** in Server mode, **Certificate…** opens the certificate tool. It inspects the certificate file, creates a self-signed one (**Create self-signed…**) or imports one from your own PKI (**Import .pfx…**, **Import PEM…**), and seals the certificate's password against this machine's master key so nobody types or sees it. Where the site runs its own authority, the tool says so and that the certificate *renews itself*. Every field and verdict is described under *The certificate tool* in the *Configuration and deployment* chapter.

A site authority that changed. A workstation trusts the site's root from the first time it joins and refuses a node that later presents a different one, because that is also what an impostor would present. The **Site authority changed** dialog lets you replace the root only after typing the new root's SHA-256 fingerprint, read on the plant server itself; see *When the site authority has changed* in the *Configuration and deployment* chapter.

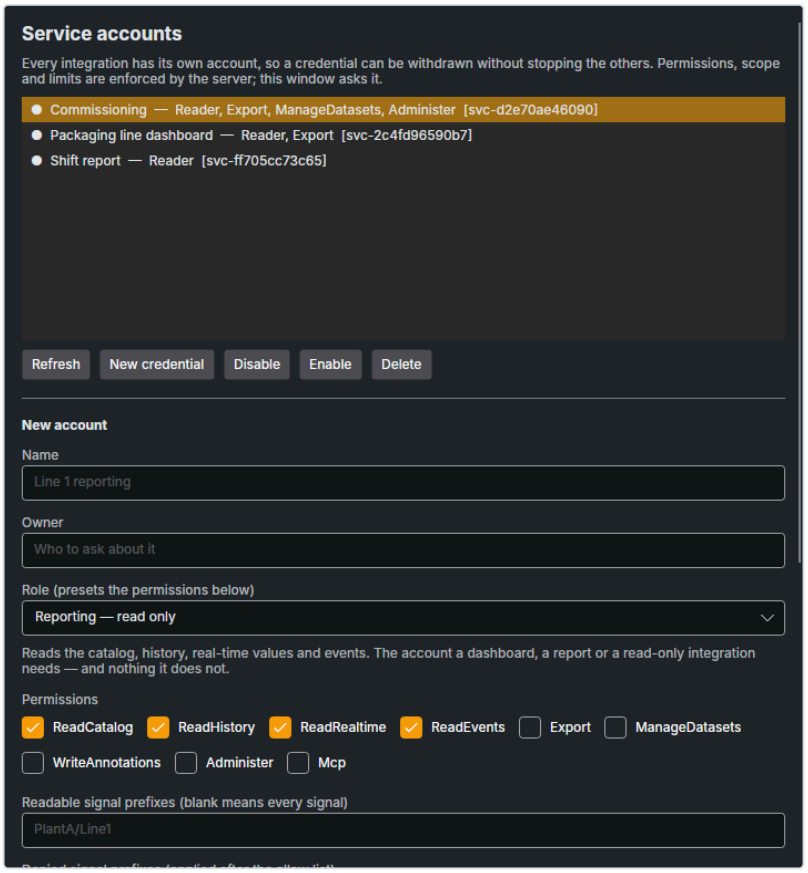
Client certificates. A server can require each workstation to present a certificate of its own; which ones it admits is managed in **Administration ▸ Trusted clients…** (see the *Configuration and deployment* chapter). Revocation is honoured at both ends: a server refuses a withdrawn client, and a client refuses a withdrawn server and checks that the name it dialled matches the certificate.

Pinning. On a plant network with no PKI, a client can pin one known self-signed server certificate under **TLS ▸ Pinned server certificate** in the Deployment screen. Pinning one exact certificate is a real check; accepting any certificate would not be.

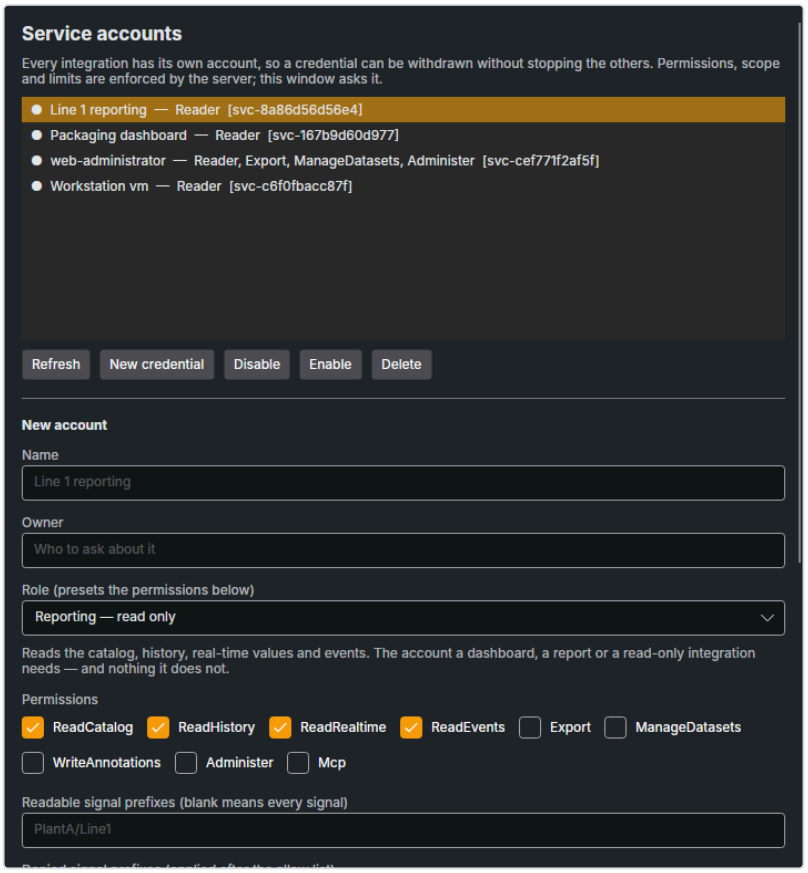
Service accounts

Integrations — a Grafana data source, a Python notebook, an MES reading production data, a scheduled report export — should never sign in as a person. **Administration ▸ Service Accounts…** (Administrator) creates a separate machine account for each one, so one credential can be withdrawn without stopping the others. Permissions, scope and limits are enforced by the node; the window only asks it.

The window asks the node **as the person signed in to this Desktop**, the same way the browser's **Administration ▸ Service accounts** does. On a Desktop joined to a node elsewhere, an Administrator therefore sees and manages the node's accounts — including the read-only one the node issued to this workstation when it joined, named *Workstation* followed by the computer's name. Somebody the node does not allow to manage users is shown no accounts and cannot create, renew, disable or delete one. The node writes the person's name, not the workstation's, in its audit trail.



Administration ▸ Service Accounts, with three accounts listed and the New account form: name, owner, role preset, permissions and signal prefixes.



Administration ▸ Service Accounts on a Desktop joined to the packaging node: the node's four accounts — Line 1 reporting (selected) and Packaging dashboard as Reader, web-administrator with Export, ManageDatasets and Administer, and the workstation's own read-only account — each with its id; Refresh, New credential, Disable, Enable and Delete; and the New account form with the Reporting — read only preset, which ticks ReadCatalog, ReadHistory, ReadRealtime and ReadEvents

Control	What it does
Account list	One row per account: its name, its permissions and its id, for example `Shift report — Reader [svc-454d6e127fa9]`
Refresh	Reads the list again
New credential	Issues a new token for the selected account; the old one stops working
Disable / **Enable**	Suspends the account without deleting it, or restores it
Delete	Destroys the account and its credential; anything using it stops at once
Name / **Owner**	What the account is for (for example `Line 1 reporting`) and who to ask about it
Role (presets the permissions below)	A preset such as *Reporting — read only*, which ticks `ReadCatalog`, `ReadHistory`, `ReadRealtime` and `ReadEvents`
Permissions	The individual permissions: `ReadCatalog`, `ReadHistory`, `ReadRealtime`, `ReadEvents`, `Export`, `ManageDatasets`, `WriteAnnotations`, `Administer`, and `Mcp` (use by an AI assistant over MCP, part of the Industrial integration module; in no preset)
Readable signal prefixes / **Denied signal prefixes**	Restrict the account to part of the plant, for example `Packaging/` allowed and `Packaging/Vision/` denied. Blank means every signal; the deny list is applied after the allow list
Credential expires after (days)	How long the token lives

Control	What it does
Create account	Creates it and shows the token

A token is shown once, when it is created or renewed: "*Copy this now. It is shown once and cannot be retrieved again — the server keeps only a hash of it.*" Use **Copy token** or **Copy connection (address + token)**, paste it where it is needed, then press **I have copied it**.

Creating and managing service accounts is part of the base product. Using them to read data from outside — the data-access server, its REST and gRPC interfaces and the Python and .NET SDKs — belongs to the Data access and SDKs module, and is described there where that module is licensed. The number of service accounts is limited by the licence.

Licensing and modules

This chapter explains what a Meelmo installation is permitted to do and how that is decided: the three editions, the licence and entitlement files, the trial, the **Help ▸ License** window, the **Administration ▸ Modules** window and the separately sold modules, client seats, and what happens when a licence or trial runs out.

Licensing never contacts anybody. There is no activation call, no inbound connection and no online check: a licence is a signed file, verified on the machine against the key the build carries. Nothing stops because a check could not reach the vendor, because there is no such check.

Editions

Editions are **different downloads**, not one download that a licence file reconfigures. The edition's name is in the package name: `Meelmo-Community-…` is the free edition; anything else is a paid one.

	Community	Standard	Professional
Price	Free, perpetual	Paid	Paid
Licence file	None — nothing to install or activate	`.meelmolic` plus entitlement file	`.meelmolic` plus entitlement file
Trial	Not applicable; nothing expires	30 days	30 days
Tags	32	256	Unlimited
Drivers	Simulator, Modbus TCP/RTU, OPC UA, MQTT, MTConnect, IEC 60870-5-104, DNP3, BACnet/IP, Tasmota, Sparkplug B, Shelly, SunSpec, LoRaWAN, IO-Link, M-Bus, KNX/IP	All	All
Recording	One file per Start/Stop	One file per Start/Stop	Also continuous rotating, retention and triggered
Recorder service (headless, unattended)	No	No	Yes
Concurrent clients (seats)	1	None — a desktop analysis tool, not a server	2 included; more can be bought
Browser HMI	Yes, one concurrent viewer	With the Web HMI module	With the Web HMI module
Historian	Embedded (SQLite)	Embedded, plus SQL Server, Oracle, TimescaleDB, QuestDB or ClickHouse with their modules	Same as Standard
Lineage and impact warnings	Yes	Yes	Yes
Tag drift	Checked on demand	On demand and on a schedule	On demand and on a schedule

The vendor drivers are absent from a Community download, not disabled in it. Siemens S7, Allen-Bradley EtherNet/IP, Beckhoff ADS, Mitsubishi MELSEC, Omron FINS, the database historians and MQTT/Sparkplug publication are not in that package. A project naming one of them opens on Community with the connection listed and refusing to start, and works as soon as it is opened on an edition that has the driver. Installing a paid licence on a Community build raises its limits but cannot add a driver.

Upgrading is a new download, not a key. Projects, recordings, screens and history carry over unchanged.

The licence

A paid licence is **two files**, and both are needed:

File	Carries
The licence, `license.meelmolic`	The licensee, the edition, the expiry date, the purchased quantities (for example the number of client seats) and the binding to this machine
The entitlement file, `entitlements.opsent`	The **capabilities**: which parts of the product, and which modules, this site may run

A module is licensed only when the entitlement file grants **every** capability that module needs. A licence installed without its entitlement file reads as licensed on every screen and runs the embedded historian and **no module at all** — less than the same machine had during its trial. If the entitlement file did not arrive, ask for it before installing either.

Binding. A licence is bound to one machine and fails on a copy. On an installed machine the binding is the **machine ID**, derived from stable machine characteristics and unchanged by a reboot. In a container, which has no stable machine identity, it is the **node ID**. A licence bound elsewhere is refused with "*This license is locked to another computer. Request a license for this machine using the ID below.*"

Where it lives. The licence and entitlement files are kept in the data folder (normally `C:\Meelmo\Data`) and are read from there by every Meelmo program on the machine — the Desktop, the recorder service and the browser host. Install a licence once and the whole machine is licensed by it. Uninstalling or upgrading leaves them in place.

The trial

With no licence, a paid edition runs a **30-day trial with every capability and every module enabled, at small quantities**:

Quantity	Trial limit
Tags	128
Concurrent clients	2
Concurrent historical queries	2
Real-time streams	2
Concurrent exports	1
Signals per query	250
Signals per subscription	100
Historian instances	1

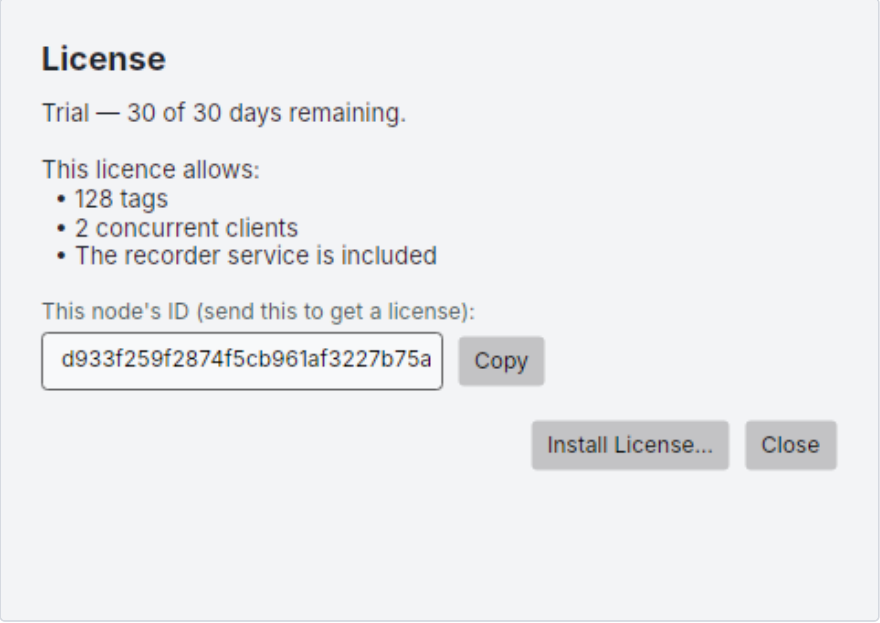
Quantity	Trial limit
Service accounts	2

The limits are chosen so that an evaluation never meets them and a production plant meets them at once. During a trial, the recorder service also runs unattended for at most **8 hours at a time**; it then stops acquisition with "*This is a trial, and a trial runs unattended for 8 hours at a time. Acquisition has stopped and will resume when the service is restarted.*"

The trial's clock only counts forwards: setting the system clock back does not extend it.

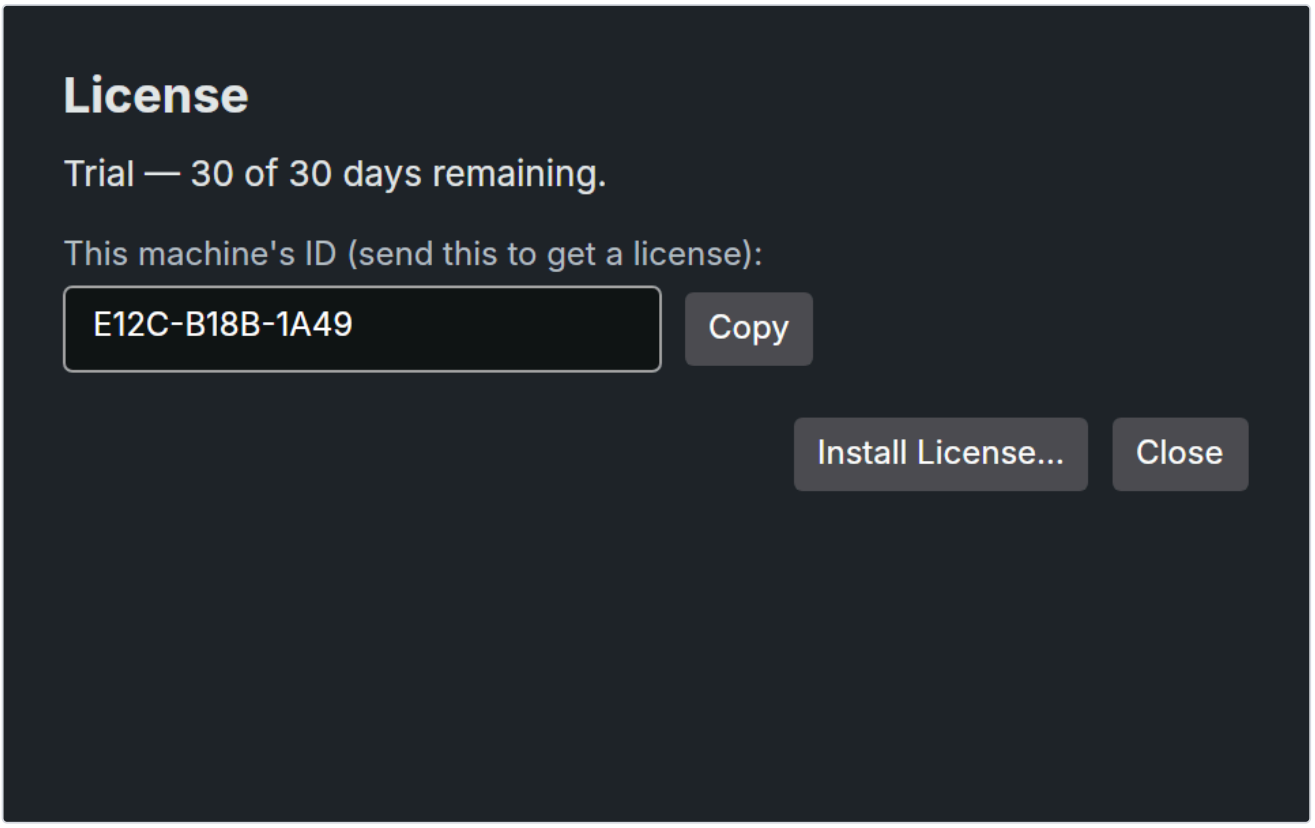
Help ▸ License

Help ▸ License… shows what this installation is permitted and installs a new licence. Anybody may open it; installing a licence needs the Administrator role.



Help ▸ License on a new installation during its trial: the days remaining, this machine's ID with a Copy button, and Install License.

Part	Meaning
Status line	The licence state. One of: *Trial — 30 of 30 days remaining.*; *Licensed to Packaging Ltd (Professional) — expires 2027-09-30.*; *Licensed to … (…).* for a licence without an expiry; *License for … expired on ….*; *Your 30-day trial has ended. Please install a license to continue.*; or, on the free edition, *Community Edition — free, no license required. Up to 32 tags.*
This licence allows:	What the licence permits in numbers: the tags (or *Unlimited*), the concurrent clients, and whether *the recorder service is included*
This machine's ID (send this to get a license):	The ID a licence must be bound to, with **Copy**. In a container the label reads **This node's ID**
Install License…	Opens a file picker for a *Meelmo licence or entitlements* file (`*.meelmolic` or `*.opsent`). The window works out which one it has been given from its contents
Close	Closes the window. When the window was opened because the application cannot start (see below), the buttons are **Continue** (while the installation is usable) and **Exit**



The same window in the dark theme: Trial — 30 of 30 days remaining, and the machine ID B820-AB89-13B0.

On a Desktop attached to a server as a client, the window shows the **node's** licence — the one on the machine that holds the node — because that is the licence acquisition runs under.

Worked example: turning the trial into a licence

The packaging line's plant server `pkg-srv-01` has run the trial for a week and the site has bought a Professional licence with the Web HMI module.

1. On the server (or on a Desktop attached to it), open **Help ▸ License…**.
2. Press **Copy** beside the machine ID, for example `B820-AB89-13B0`, and send it to your supplier. The product never transmits it; you do.
3. You receive two files: `license.meelmolic` and `entitlements.opsent`.
4. Sign in as an administrator, open **Help ▸ License…** and press **Install License…**. Choose `license.meelmolic`.
5. Press **Install License…** again and choose `entitlements.opsent`.
6. The status line now reads *Licensed to Packaging Ltd (Professional) — expires …*, and **This licence allows:** lists *Unlimited tags*, the number of concurrent clients bought and *The recorder service is included*.
7. Open **Administration ▸ Modules…**, check that **Web HMI** now shows as licensed, and enable it (see *Enabling a module*).

Both installations are recorded in the node's audit trail.

With the Web HMI module, **Administration ▸ License** in the browser shows the same information and installs either file.

Modules

Beyond the base product, Meelmo sells separate **modules**. The base product is everything a site needs to acquire and run a plant: the Desktop, the embedded node, acquisition and the standard drivers, the Simulator, trends, alarms and recording, the embedded (SQLite) historian, configuration, roles and permissions, backup and restore, the commissioning report, detection rules, the semantic model, datasets and reports. None of these is ever a module, so a licence for Meelmo always lets a site acquire and record.

Module (id)	Adds	Capabilities the entitlement file must grant
Web HMI (`web-hmi`)	The browser operator runtime and the host that serves it: operator screens, the HMI Studio, the browser's report and analysis screens, automation, live widgets and gated command. Metered by concurrent real-time streams	`HistoricalApi` , `RealtimeApi`
Data access and SDKs (`data-access-sdk`)	The data-access server, its REST and gRPC interfaces, and the Python and .NET SDKs that read them	`DataAccessServer` , `HistoricalApi` , `RealtimeApi` , `PythonSdk` , `DotNetSdk`
SQL Server historian (`historian-sqlserver`)	Writing this node's history to Microsoft SQL Server	`SqlServerProvider`
Oracle historian (`historian-oracle`)	Writing this node's history to Oracle	`OracleProvider`
TimescaleDB historian (`historian-timescale`)	Writing this node's history to TimescaleDB	`TimescaleProvider`
QuestDB historian (`historian-questdb`)	Writing this node's history to QuestDB	`QuestDbProvider`
ClickHouse historian (`historian-clickhouse`)	Writing this node's history to ClickHouse	`ClickHouseProvider`
Industrial integration (`industrial-integration`)	Publishing to MQTT, as a Sparkplug B edge node (unified namespace) and to Kafka; the node's governed MQTT broker and read-only OPC UA server; and the MCP surface that AI assistants query	`MqttPublishing` , `SparkplugPublishing` , `McpServer` . Publishing anywhere also needs `EnterpriseStreaming` ; Kafka, the broker and the OPC UA server go with `SparkplugPublishing`
Models (`models`)	Fitting models to recorded history, and estimating from them what the plant does not measure (soft sensors and forecasts)	`Models` , `HistoricalApi`

The **Studio Assistant** — the authoring assistant inside the browser's Report, HMI and automation Studios — is sold separately as well, but it is a licence **capability** (`StudioAssistant`) rather than a module: it appears in the entitlement file, not in **Administration ▸ Modules**, and it is switched on and given a model provider in its own settings. It works inside the browser's Studios, so it is only useful together with the Web HMI module.

A module that is not licensed is not hidden. Its screens and settings are simply unavailable, and the Modules window says what it would need.

The six states of a module

Whether a module runs is decided on six axes, in this order. The first one that fails is the one reported, because telling somebody their licence does not cover a module they have not installed sends them to buy something they already own.

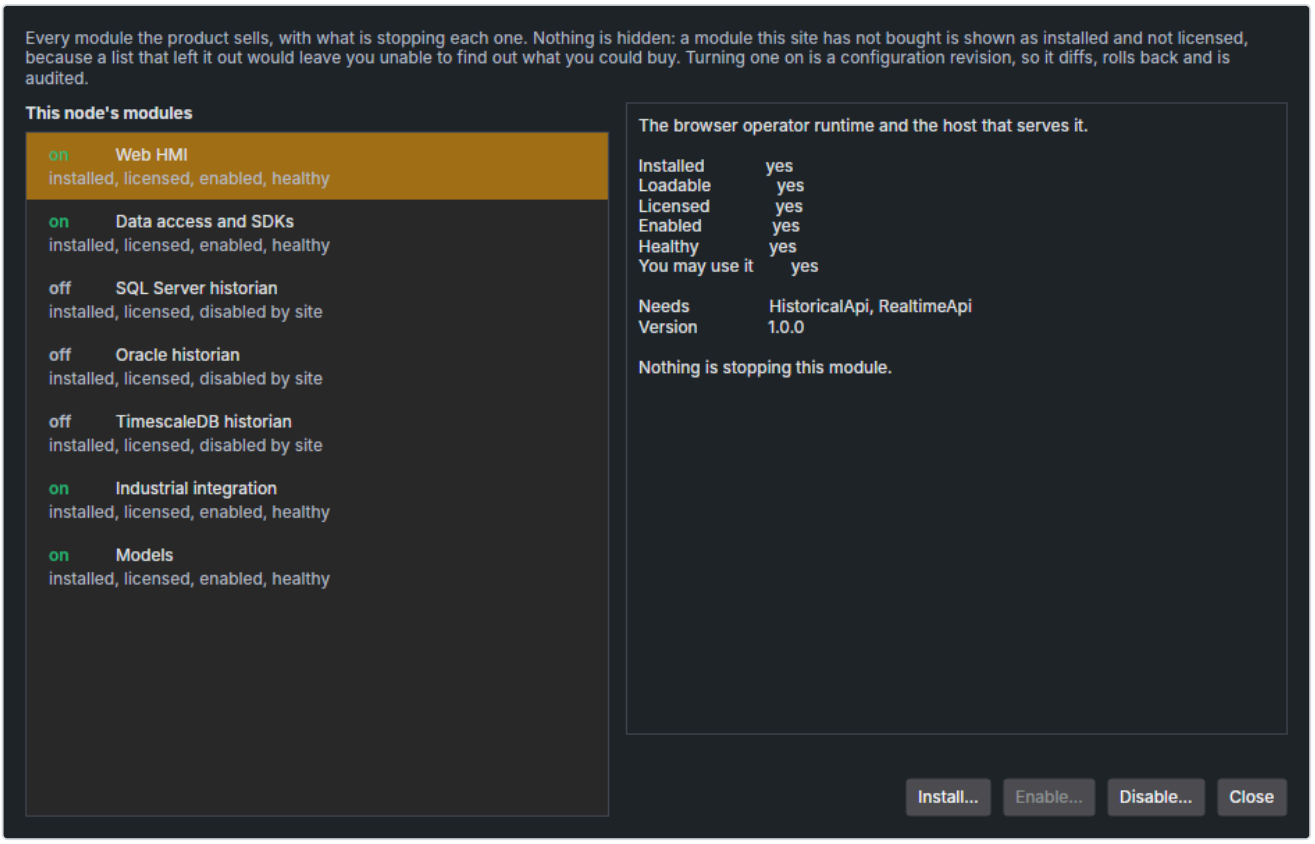
Axis (as the window labels it)	Question	When it fails
Installed	Is the module present on this node?	*Not installed on this node. Install it before turning it on.*
Loadable	Can this build load it (version and signature)?	*Installed, and this build cannot load it. Upgrade Meelmo, or reinstall the module.*
Licensed	Does the entitlement file grant all of its capabilities?	*This installation's licence does not include it. It stays visible so you can see what it would need.*
Enabled	Has this site turned it on?	*Licensed, and this site has not turned it on. That is a decision, not a fault.*
Healthy	Is it working now?	*Enabled, and not working. The detail above says what the node reported.*
You may use it	Does the signed-in person have permission?	*You do not have permission to use this module. That says nothing about the module.*

When nothing fails, the window says *Nothing is stopping this module.*

Being licensed and being enabled are deliberately separate. Buying a module does not switch it on; turning it on is a separate, recorded decision by the site.

Administration ▸ Modules

Administration ▸ Modules… (Administrator) lists every module the product sells, with what is stopping each one. Its title bar reads *Modules and licence*.



Administration ▸ Modules: the list of this node's modules on the left, the per-axis detail of the selected module on the right, and the Install, Enable and Disable buttons.

Part	Meaning
This node's modules (left)	One row per module: **on** or **off**, its name, and a one-line summary. **off** is grey when the module is simply not licensed or not enabled, and highlighted when something else is stopping it
Detail (right)	For the selected module, one line per axis — **Installed**, **Loadable**, **Licensed**, **Enabled**, **Healthy**, **You may use it** — each *yes* or *no*; **Needs** (the capabilities it requires); **Version**; any detail the node reported; and the sentence for the first failing axis
Install…	Installs a module delivered later as a signed *Module manifest* (`module.json`)
Enable…	Turns the selected module on. Offered only when it is installed, loadable, licensed and you may use it
Disable…	Turns the selected module off. Offered whenever it is on
Close	Closes the window

Enabling and disabling are **configuration revisions** in the node's `modules` section: they appear in **Acquisition ▸ Node history…**, can be rolled back and are audited like every other change.

Disabling is never refused, for any reason — not by the licence, a signature or a version — because the way back out of a configuration must always be open. Anything running on a disabled module stops at the next start.

On an installation without a node the window says so: modules belong to a node.

Installing a module bought later

A module bought after installation arrives as a signed manifest file.

1. Open **Administration ▸ Modules…** and press **Install…**.
2. Choose the `module.json` file. The node verifies its signature against the key the build carries and refuses it, with the reason, if it does not hold.
3. The window reports "*Installed '…'. It is not enabled yet.*" The module is stored beside the node, so it survives an application upgrade and follows the node if its database moves.
4. If the licence does not yet include it, install the new entitlement file from **Help ▸ License…**.
5. Enable it as below.

The audit trail records who installed the module and, separately, who chose to run it.

Enabling a module

Example: enabling the TimescaleDB historian on the packaging line's node, whose new entitlement file grants `TimescaleProvider`.

1. Open **Administration ▸ Modules…** and select **TimescaleDB historian**.
2. Read the detail: **Installed** *yes*, **Loadable** *yes*, **Licensed** *yes*, **Enabled** *no*, and *Licensed, and this site has not turned it on.*
3. Press **Enable…** and confirm "*Enable TimescaleDB historian? This is a configuration revision: it diffs, rolls back and is audited like every other change.*"
4. The window reports *TimescaleDB historian is enabled.* and the row shows **on**.
5. Configure the historian itself in **Data ▸ Historian…**: a module's own settings live in the section that owns them, not in the Modules window.

If **Enable…** is greyed out, the detail says which axis is stopping it. If the node refuses, its own message is shown in red; for a licence it names what the module needs.

Seats

A **seat** is one concurrent client of a server: a Desktop attached in Client mode, or a browser viewer where the Web HMI module is used. The licence says how many there are, and **This licence allows:** in **Help ▸ License…** shows the number.

Edition	Concurrent clients
Community	1
Standard	0 — Standard does not serve clients
Professional	2 included; more are bought as part of the licence
Trial	2

A standalone installation uses no seat. A client takes a seat when it attaches, shows it in the status bar (for example *seat 1 of 2*) and gives it back when it signs out or closes. Seats are 30-second leases that the client renews while connected, so a workstation that crashes returns its seat on its own. When every seat is taken, a new client is refused with a message such as *All 2 licensed client seats are in use*.

Other quantities in a licence — concurrent historical queries, real-time streams, exports, signals per query and per subscription, historian instances and service accounts — are enforced by the node and its servers in the same way: a request beyond the limit is refused with a message saying so.

The tag limit is checked when acquisition starts on the machine that holds the node (a client does not count the server's tags against its own licence). A project enabling more tags than the edition allows is refused with "*The Standard edition is limited to 256 tags; this project enables 300. Disable tags or upgrade the license.*"

When a licence or trial ends

What happens depends on where you look, and the rule behind all of it is the same: **a licence decides what may be started; it does not pull the plug on something already running, and it never blocks the way back.**

Situation	What happens
The Desktop starts with an expired licence or an ended trial	Before anything else, **Help ▸ License** opens with the reason (*License for … expired on …* or *Your 30-day trial has ended*), the machine ID, **Install License…** and **Exit**. Installing a valid licence continues into the application; otherwise it closes
The recorder service starts unlicensed	It refuses to start and writes the reason and the machine ID to its log. A plant server needs its licence installed before it is left unattended
A trial ends	The trial grants nothing new. Modules and capabilities it allowed are no longer licensed
A licensed module is no longer covered	It shows **Licensed** *no* in **Administration ▸ Modules**. It cannot be enabled or started; it can always be disabled
Trial recorder service after 8 hours unattended	Acquisition stops until the service is restarted (see *The trial*)

Nothing is deleted when a licence ends. The node's configuration, revisions, accounts, audit trail, recordings and history remain; installing a valid licence and entitlement file brings the same installation back exactly as it was.

The Community Edition has no licence and never expires; none of the above applies to it.

Diagnostics and support

This chapter describes the tools that tell you whether an installation is healthy and that produce the evidence to keep or to send when it is not: the live acquisition diagnostics, the commissioning report, the node's configuration history, the support bundle, the **Help** menu (the manual, **About Meelmo…**, checking for updates) and where the logs and other files are kept.

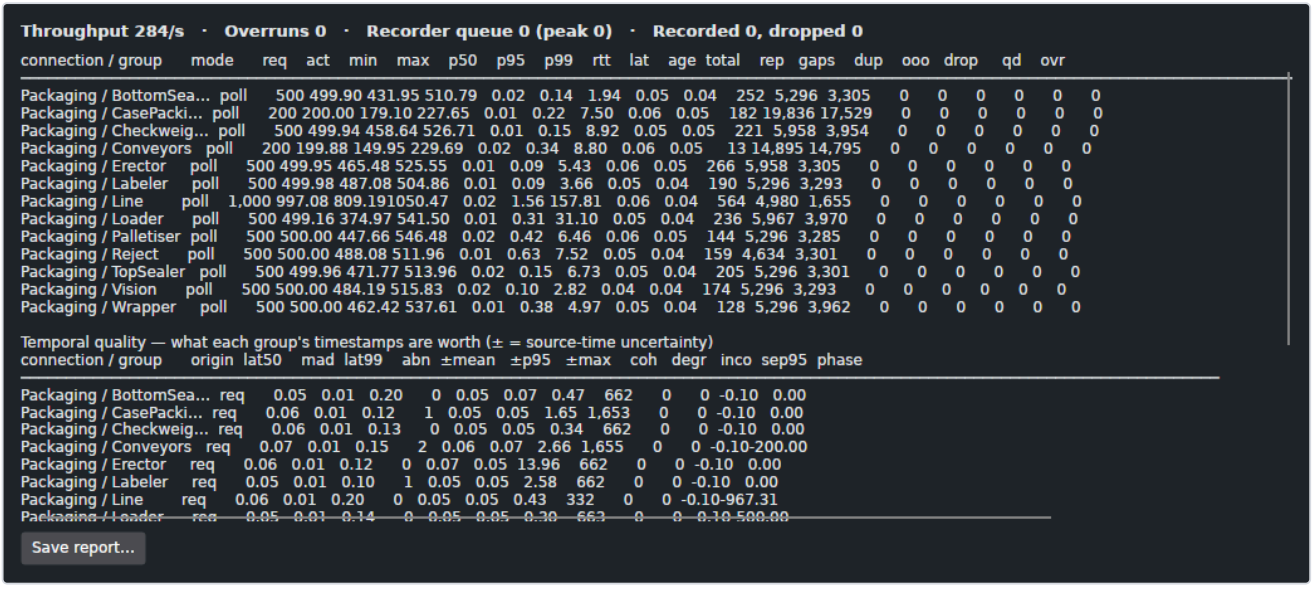
Most of these screens answer one of three questions:

Question	Screen
Is acquisition sampling the way it was configured, right now?	**Administration ▸ Diagnostics…**
Is this node ready to hand over, by its own account?	**Administration ▸ Commissioning report…**
What changed in the configuration, who changed it, and can it be undone?	**Acquisition ▸ Node history…**

For the timing of a finished recording, see **Recording ▸ Recording Timing Analysis…** and **File ▸ Export Timing Evidence (CSV)…** in the *Recording and playback* chapter. For the audit trail, see the *Users, roles and security* chapter.

Acquisition diagnostics

Administration ▸ Diagnostics… opens *Acquisition Diagnostics*, the live view of acquisition health. It refreshes several times a second while acquisition runs. Open it before you believe a trend: a group that is not sampling the way it was configured is found here at commissioning, or months later during an investigation, when the recordings are already wrong.



The Acquisition Diagnostics window while the packaging line acquires: about 284 samples a second, one row per acquisition group with its mode, requested and actual period, latency percentiles and sample counts, then the temporal-quality table and the coherence frames

The summary line

`Throughput … /s · Overruns … · Recorder queue … (peak …) · Recorded …, dropped …`

Item	Meaning
Throughput	Samples acquired per second, across every connection.
Overruns	Polls that could not start on time because the previous one had not finished.
Recorder queue (**peak**)	Samples waiting to be written to the recording, now and at the most.
Recorded, **dropped**	Samples written to the recording, and samples dropped by best-effort recording.

The acquisition table

One row per connection and acquisition group (`connection / group` , for example `Packaging / Checkweigher`):

Column	Meaning
mode	`poll` (polling), `sub` (subscription) or `cyc` (cyclic stream).
req	The requested period, in ms.
act	The actual mean period (or the period the device revised it to), in ms.
min, **max**	The shortest and longest actual period.
p50, **p95**, **p99**	Jitter percentiles, in ms.
rtt	Read round-trip time.
lat	Source-to-receive latency.
age	How old the source value was when read.
total	Samples received.
rep	Repeated samples (the same value delivered again).
gaps	Sequence gaps.
dup	Duplicate packets.
ooo	Out-of-order packets.
drop	Dropped samples.
qd	Driver queue depth.
ovr	Poll overruns for this group.

A value that cannot be measured is shown as a dash, not as zero.

Temporal quality

Below the first table, *Temporal quality — what each group's timestamps are worth (± = source-time uncertainty)*:

Column	Meaning
origin	Where the source time comes from: `req` (the request interval), `dev` (a device timestamp), `srv` (a server timestamp), `arr` (stream arrival) or `rcv` (client receive).
lat50, **mad**, **lat99**	Latency median, median absolute deviation, and 99th percentile.

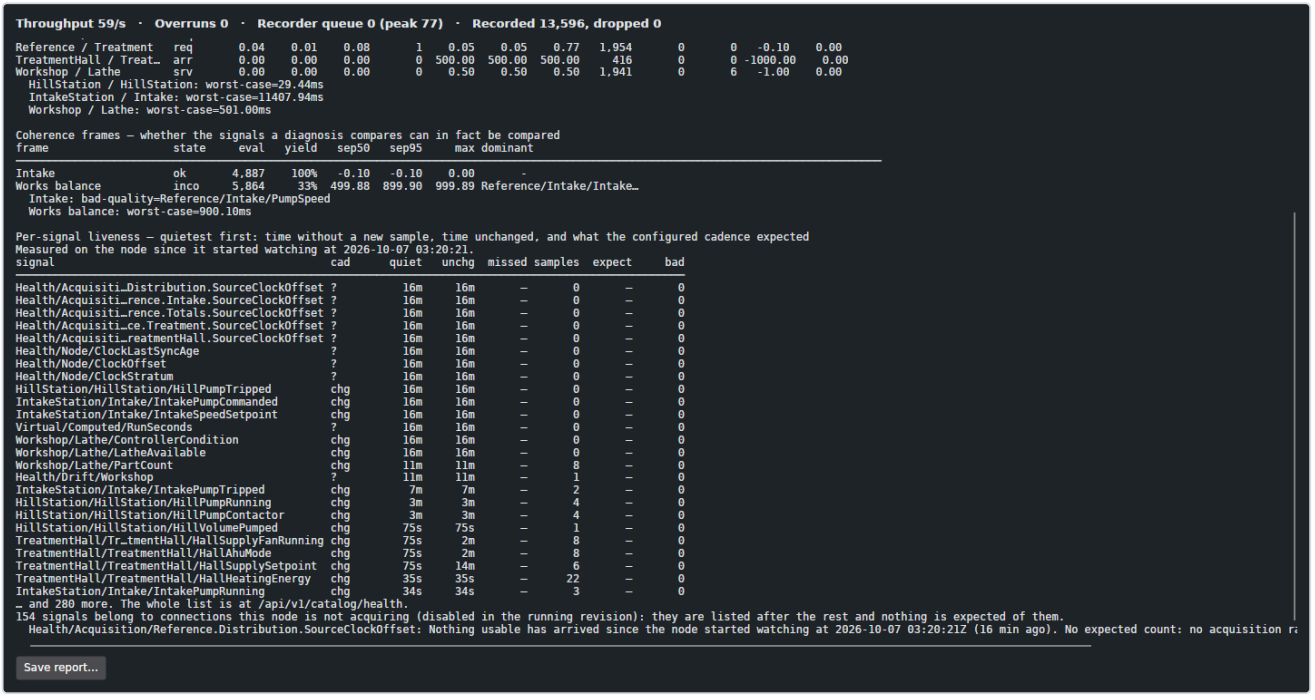
Column	Meaning
abn	Abnormal responses.
±mean, **±p95**, **±max**	The source-time uncertainty.
coh, **degr**, **inco**	Batches judged coherent, degraded within bounds, and incoherent.
sep95	95th-percentile separation between signals in a batch.
phase	Estimated phase shift.

Where the node has cross-source *coherence frames* configured, a third table, *Coherence frames — whether the signals a diagnosis compares can in fact be compared*, shows each frame's **state** (`ok` , `degr` , `inco`), evaluations, **yield**, separation percentiles and the dominant signal, with the reason for the last verdict.

Per-signal liveness

At the foot of the window, *Per-signal liveness — quietest first*: the node's own measure of each signal since it started watching, the 25 quietest first. **cad** is the cadence its acquisition group was asked for (`poll` , or `?` where nothing paces it), **quiet** the time without a new sample, **unchg** the time unchanged, **missed** the whole polling periods in which no sample arrived, **samples** and **expect** what came and what the cadence promised, and **bad** the bad readings. A signal id too long for its column is cut in the middle, so its end — the part that tells it from its neighbours — stays. The line under the table says the quietest one in words. The whole list is `GET /api/v1/catalog/health` .

The signals of a connection this node is not acquiring — disabled in the revision it runs — are not counted as missing. They come after every signal that is acquired, with **cad** `off` and a dash under **missed** and **expect**, and a line under the table says how many there are. A connection somebody switched off is not a feed that went quiet.



Per-signal liveness on the reference works: the signals the node acquires, quietest first, then those of the connections it does not acquire, marked off

Save report… writes everything on the screen, with a timestamp and the estimator version, to `acquisition-diagnostics-YYYYMMDD-HHMMSS.txt`.

Worked example: checking the packaging line's scan rates

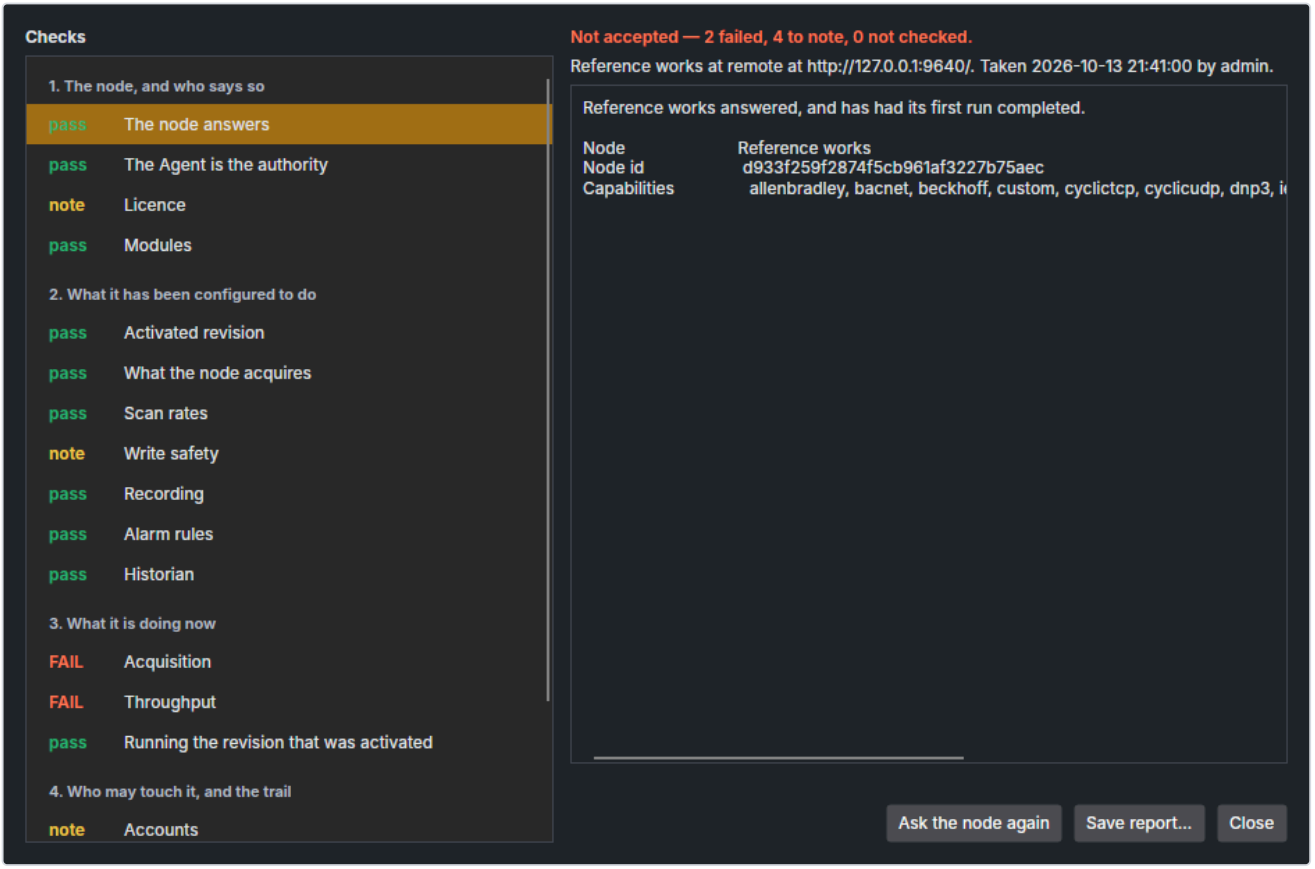
The demonstration polls the ten machine groups every 500 ms, `Conveyors` and `CasePacking` every 200 ms, and `Line` every 1000 ms.

1. Start acquisition (`F5`) and open **Administration ▸ Diagnostics…**.
2. Check that **req** reads `500` for `Packaging / Erector` and **act** is close to it. On a general-purpose operating system polling is best effort, so a few milliseconds of difference is normal; **p99** tells you how bad the worst polls are.
3. Check **ovr** is `0` for every group. If `Packaging / Conveyors` showed overruns, the controller could not answer at 200 ms; slow the group down or split it (see the *Acquisition and I/O configuration* chapter).
4. Check **gaps** and **drop** stay at `0`.
5. Click **Save report…** and file the text with the commissioning records.

The commissioning report

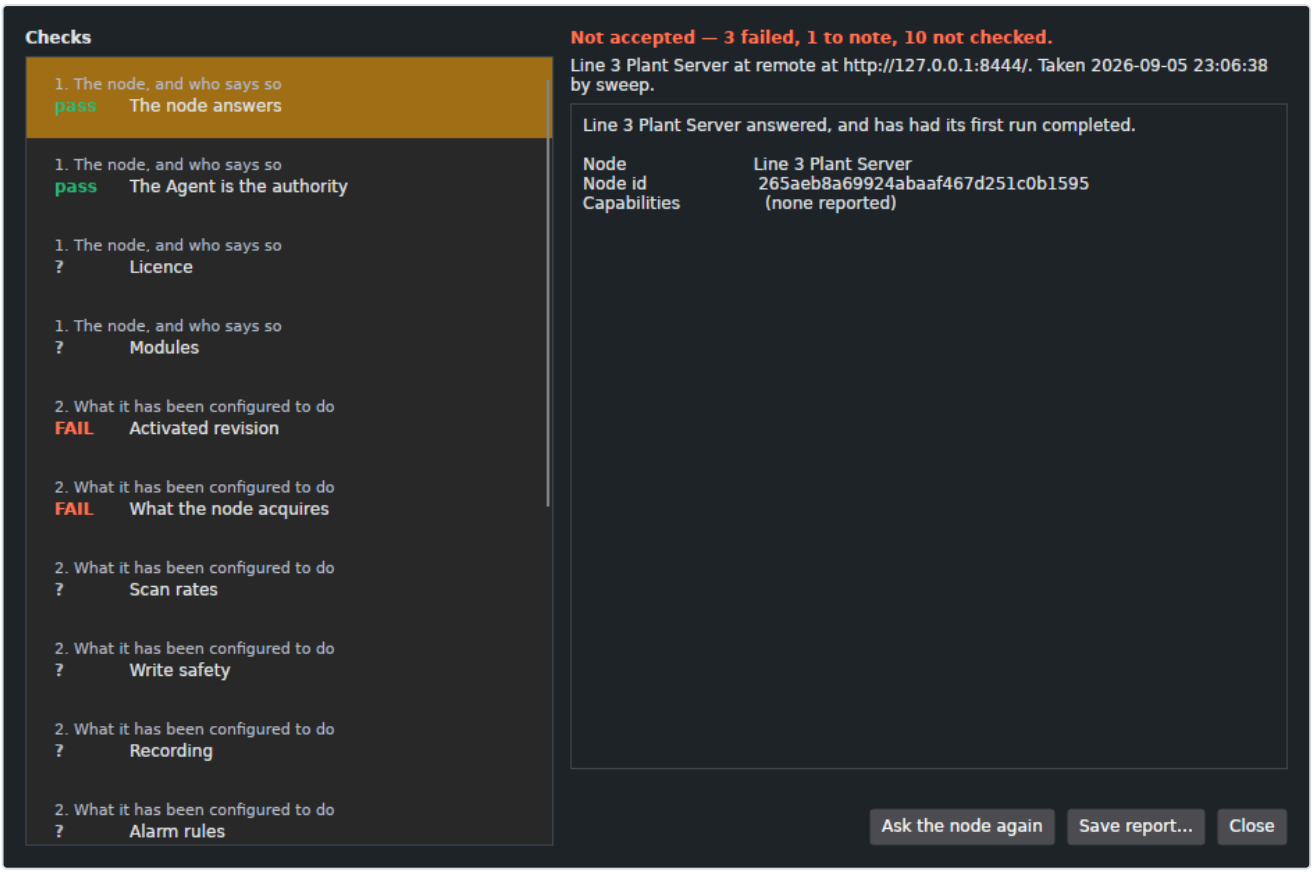
Administration ▸ Commissioning report… asks the node a series of questions about itself and produces the document to hand over at the end of a commissioning: what is configured, what it is doing, who may touch it, and whether it can be recovered. Every line is read from the node through the node's own operations; nothing is read from a file on the workstation. An installation without a node says *This installation has no node, so there is nothing to commission.*

While it works, the window says *Asking the node…*.



The Commissioning report window while it is asking the node: the empty Checks list on the left, the detail pane on the right, and the Ask the node again, Save report and Close buttons.

When it has answered, the heading gives the verdict, *Accepted — nothing failed, … to note, … not checked.* or *Not accepted — … failed, … to note, … not checked.*, followed by the node's name and address, when the report was taken and by whom.



A completed commissioning report for a plant server that is not accepted: 3 checks failed, 1 to note and 10 not checked. The first check, The node answers, is selected and shows the node's name and id.

The **Checks** list on the left shows every check with its verdict:

Verdict	Meaning
pass	The node satisfied the check.
note	Satisfied, with something to be aware of (for example a licence expiring soon, or a trial).
FAIL	Not satisfied. Any failure makes the report *Not accepted*.
n/a	Does not apply to this node.
?	Not checked: the node could not be asked, or did not answer that question.

Select a check to see, on the right, what the node said: a sentence and the facts behind it (for example **Node**, **Node id**, **Capabilities**). *Select a check to see what the node said.* is shown until you do.

The checks are grouped in five stages:

Stage	Checks
1. The node, and who says so	The node answers; The Agent is the authority; Licence (state, licensee, edition, expiry, days remaining, tag limit, client seats, recorder service, machine); Modules
2. What it has been configured to do	Activated revision; What the node acquires (connections, enabled tags, groups, virtual signals, semantic model, models); Scan rates; Write safety; Recording; Alarm rules; Historian

Stage	Checks
3. What it is doing now	Acquisition; Throughput; Running the revision that was activated
4. Who may touch it, and the trail	Accounts; Audit trail
5. Getting it back	A backup has been taken

Some failures are deliberate reminders. A node that has never had a configuration activated fails **Activated revision**; a node that still carries only the example line it was commissioned with (*Example line (simulated)*) fails **What the node acquires**, which is the reminder you want before a handover.

Button	What it does
Ask the node again	Runs every check again, for example after fixing a failure.
Save report…	Writes the report as text to `commissioning-report-YYYYMMDD-HHMMSS.txt` : node, node id, Agent, when and by whom it was taken, the verdict, and every stage and check with its facts.
Close	Closes the window.

Taking a report is itself recorded in the audit trail, with the node's name, the verdict and the number of failures.

Worked example: accepting the packaging line

1. Commission the line: connections, groups and tags configured and activated, the historian configured, alarm rules in place, accounts created, and a backup taken with **Administration ▸ Back up / restore node…**.
2. Start acquisition and let it run for a few minutes, so that **Throughput** has something to measure.
3. Open **Administration ▸ Commissioning report…**.
4. Work through every **FAIL** and **?**. For example, *A backup has been taken* fails until a backup exists; **Licence** shows **note** while the node runs on a trial.
5. Click **Ask the node again** until the heading reads *Accepted*.
6. Click **Save report…** and hand the text file over with the diagnostics report from the previous section.

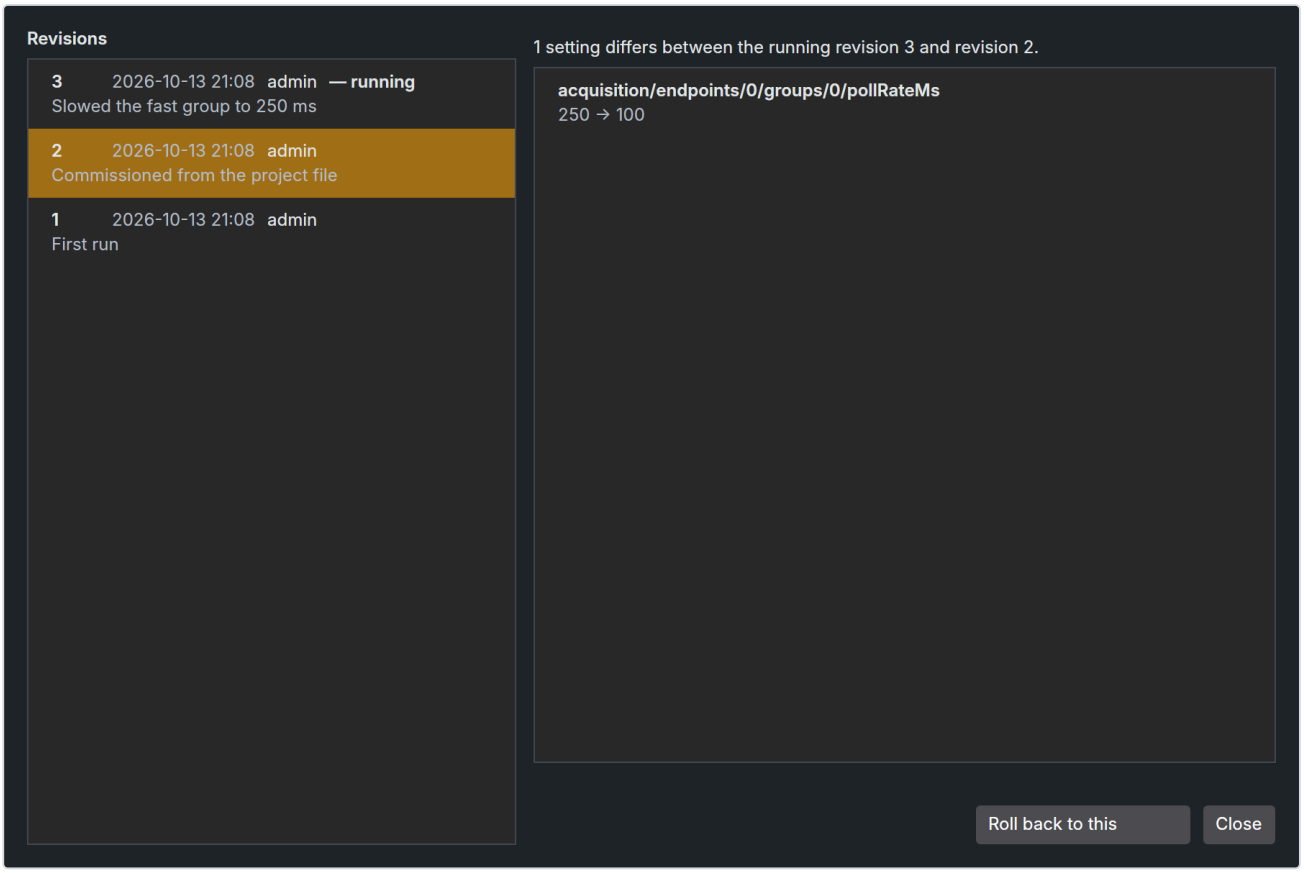
Node history: configuration revisions

Acquisition ▸ Node history… shows the node's configuration as a sequence of revisions: what changed, who activated it and when. Every change that is saved to the node (I/O configuration, virtual signals, alarm rules, recording settings, modules) is a new revision.



Acquisition ▸ Node history on the packaging node: fourteen revisions, newest first, each with its number, time, account and comment (the demonstration datasets and reports saved, then its automation rules). Revision 13 is selected, so the right-hand side says one setting differs from the running revision 14 and shows the automation rules it would restore; Roll back to this is below

The **Revisions** list shows, newest first, each revision's number, when it was activated, by whom, and its comment. The running revision is marked **— running**. Select another revision to compare it with the running one: the right-hand pane says *1 setting differs between the running revision 3 and revision 2.* (or *Revision … is identical to the running revision …*) and lists each setting that differs, as the setting's path and *old* → *new*, or *Added:* / *Removed:*.



Node history with three revisions: revision 3 is running, and revision 2 is selected. One setting differs, the poll rate of the first acquisition group, 250 in the running revision against 100 in revision 2.

Roll back to this makes the selected revision the running one. It asks first: *Roll back to revision …? The node will restart acquisition on it. This is recorded as a new revision, so it can itself be rolled back.* A rollback is an ordinary activation of an older configuration: it is audited, and it takes effect on every screen the node serves.

An installation that keeps its configuration in a project file rather than a node has no history; the window says so and suggests making it a node through the deployment settings (**Administration ▸ Deployment…**).

Worked example: undoing a scan-rate change

An engineer changed the `Conveyors` group from 200 ms to 100 ms, and **Diagnostics** now shows overruns.

1. Open **Acquisition ▸ Node history…**.
2. Select the revision before the change. The right-hand pane shows the poll rate `100 → 200` for that group.
3. Click **Roll back to this** and confirm.
4. The node restarts acquisition on the older configuration, and a new revision (recording the rollback) appears at the top of the list.

The support bundle

Help ▸ Save Support Bundle… collects what support needs into one zip file, named `meelmo-support-YYYYMMDD-HHMMSS.zip` by default. It contains:

Item	Content
`environment.txt`	Product and version, operating system, runtime, machine name, machine id, culture, time, and the licence state.
`logs/`	The application's log files from the last 7 days.
`settings.json`	The application's settings.

Credentials are masked: logs are scrubbed when they are written and again on the way into the bundle. The bundle deliberately does **not** contain your projects or recordings, the user store (password hashes), the audit directory, or the key store. Read it before sending it; you know your plant.

When the file is saved the status line reads *Support bundle saved: …*.

Evidence to keep and to send

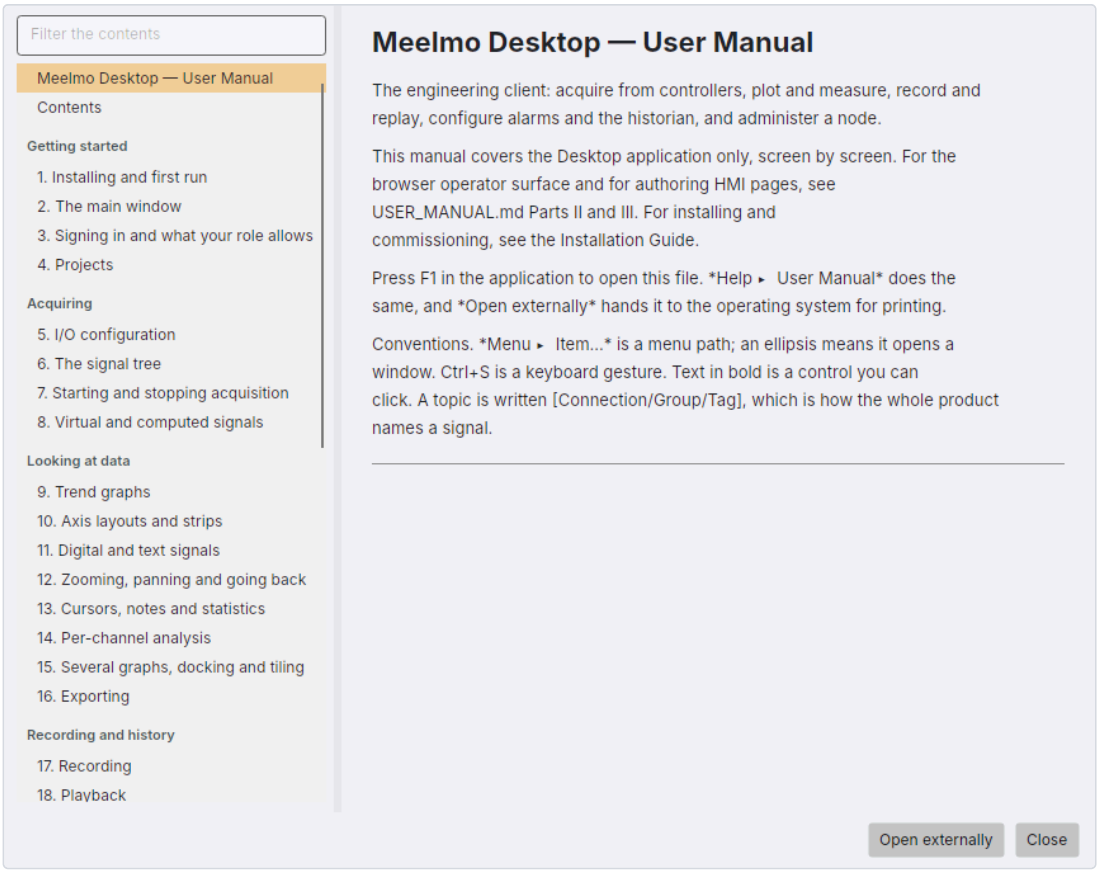
Several screens save a file that is worth keeping with a commissioning record or attaching to a support request:

File	From	Proves
`acquisition-diagnostics-….txt`	**Administration ▸ Diagnostics… ▸ Save report…**	How acquisition was sampling at that moment.
`commissioning-report-….txt`	**Administration ▸ Commissioning report… ▸ Save report…**	What the node said about itself, check by check.
Timing analysis report	**Recording ▸ Recording Timing Analysis… ▸ Save report…**	What a recording's timestamps are worth.
`<recording>-timing.csv`	**File ▸ Export Timing Evidence (CSV)…** (in playback)	Per-sample timing bounds.
A project with notes	Chart notes (**Annotations ▸ Add note here…**) and **File ▸ Save Project**	The moment you mean, marked on the chart.
`meelmo-support-….zip`	**Help ▸ Save Support Bundle...**	Logs, environment and settings.

When something looks wrong and you want to report it precisely: put a note on the chart at the moment, save the project, and send it with a support bundle.

The Help menu

Item	What it does
User Manual (`F1`)	Opens the manual inside the application.
Demonstration…	Sets up the demonstration line (see the *Getting started* chapter).
Open Logs Folder	Opens the folder the application writes its logs to.
Save Support Bundle...	See above.
Check for Updates...	See below.
License…	Shows and installs the licence (see the *Licensing and modules* chapter).
About Meelmo…	Shows the version and where settings and logs are kept.

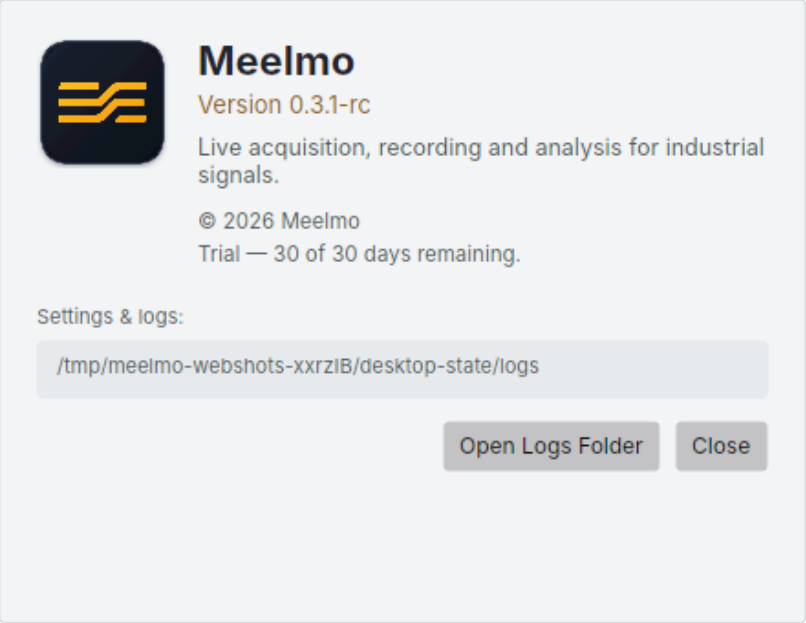


The User Manual window opened with F1: a contents list with a filter box on the left, the manual on the right, and Open externally and Close buttons.

In the manual window, **Filter the contents** narrows the contents list, and **Open externally** hands the file to the operating system, for printing for example.

About Meelmo

Help ▸ About Meelmo... shows the product name, its version (`0.4.2-rc` for this release), the tagline *Live acquisition, recording and analysis for industrial signals.*, the copyright line, and the licence state (for example *Trial — 30 of 30 days remaining.*). Under **Settings & logs** it shows the folder the logs are written to; **Open Logs Folder** opens it.



The About Meelmo window of a release-candidate build, version 0.3.1-rc, a trial with 30 of 30 days remaining, and the settings and logs folder with an Open Logs Folder button.

Quote the version from this window, and the edition and expiry from **Help ▸ License…**, whenever you report a problem.

Checking for updates

Help ▸ Check for Updates… asks the installation's update feed whether a newer release exists. It never installs anything behind you.

Answer	Meaning
No update feed is configured for this installation.	No feed address is set for this installation. Ask your supplier how updates are distributed.
You are running the latest version (…).	Nothing newer is published.
Version … is available.	Shows the release notes and, where the feed gives one, the expected SHA-256 of a publisher-signed package. You are then asked *Download and verify the update now? (No opens the download page in your browser.)*

If you choose to download, you pick where to save the file; the status line reads *Downloading and verifying update…*, and then *Update verified and saved to … (not run). Verify and install it manually.* The folder is opened for you. A package that fails verification is not saved: *The update could not be verified and was not saved: …*. Run the installer yourself, at a time that suits the plant.

When a feed is configured, the application also checks quietly at start and, if a newer version exists, says *Update available: version … (Help > Check for Updates)* on the status line.

Logs and file locations

Everything an installation owns (the node, its licence, accounts, audit log, sealed secrets, and unless the historian points at a database, its history) lives in one **state directory**, shared by the Desktop, the recorder service and the other programs of the installation. It is chosen at installation (`C:\Meelmo\Data` in

a standard Windows installation) and can be set with the `MEELMO_STATE` environment variable. It is deliberately outside the program folder, so an upgrade or an uninstall replaces the program and leaves the site commissioned.

What	Where
Application logs	`<state>\logs\meelmo-YYYYMMDD.log`, one file a day, kept 14 days. **Help ▸ Open Logs Folder** and **About Meelmo…** show the folder. Credential-shaped text is masked before it is written.
Application settings	`<state>\settings.json`
Projects	`<state>\projects\` is where *Open** and **Save** start, unless a project is already open elsewhere. Project files are `.meelmoproj`.
Recordings	`<state>\recordings\` by default (see *Recording settings…**). Files are `.opslog`.
Recorder service configuration	`recorder-config.json` in the data folder.
Audit log	`<state>\audit\audit-YYYYMMDD.jsonl`, one hash-chained file per day.
Sealed secrets	`<state>\secrets\`, including `master.key`.
Analysis presets	`*.opsanalysis`, wherever you save them.
The program	The installation folder (under `Program Files` on Windows), with the example projects and the manual.

Back up the state directory whole, including `secrets\master.key`. Administration ▸ Back up / restore node… takes the node's configuration revisions, accounts, licence, historian settings and audit log as one archive. Secrets are sealed with a key bound to the machine, so a restore onto different hardware needs them re-entered. Rehearse the restore before you depend on it.

What to include in a support request

1. The product version from **Help ▸ About Meelmo…**, and the edition and expiry from **Help ▸ License…**.
2. The deployment shape: standalone, server or client (**Administration ▸ Deployment…**).
3. A support bundle (**Help ▸ Save Support Bundle…**).
4. For anything about acquisition timing, a saved diagnostics report; for a commissioning question, the saved commissioning report.
5. For anything about a chart, the project with a note at the moment in question, and what the status bar said at the time.

Installation, setup and the programs

This chapter covers putting Meelmo on a machine and running its programs:

- the **Meelmo Setup** wizard, page by page, and what each answer does;
- coming back to it to repair, change or remove an installation;
- the same wizard served as a browser page on a server with no screen;
- installing many machines unattended from an answer file;
- the setup command line;
- the Windows installer;
- Linux paths and services;
- containers;
- the Community Edge virtual machine;
- the command-line options of the recorder service, the browser host and the Desktop;
- a redundant pair of recorders, and the tool that says which machine of a pair holds the plant.

What happens *after* installation is covered elsewhere. Moving a machine between roles and attaching workstations are in the *Configuration and deployment* chapter. Accounts are in *Users, roles and security*, and licences in *Licensing and modules*.

The examples use the demonstration packaging line. Its plant server is called `pkg-srv-01` and its node `PKG-SRV-01`.

You want to	Read
Choose a download and a role	Before you install
Install one machine	The Meelmo Setup wizard
Install a server that has no screen	A server with no screen
Install fifty workstations the same way	Answer files and unattended installation
Repair, upgrade or remove	Repair, change, upgrade and remove
Script around setup, or read what a machine has	The setup command line
Run Meelmo in Docker	Containers
Set up the free virtual machine	The Community Edge VM
Start the recorder or browser host yourself	Running the programs by hand
Keep acquiring when the recorder process dies	A redundant pair

Before you install

Which file

File	What it is
`Meelmo-<version>-setup.exe`	The Windows installer.
`Meelmo-Setup-<version>-win-x64.exe`	Meelmo Setup for Windows (preview). It is the same wizard Linux uses and installs the same components into the same places. It also takes over an installation made by the file above as an upgrade. Use whichever of the two your vendor names for the release.

File	What it is
`Meelmo-Setup-<version>-linux-x64.run`	Meelmo Setup for Linux: the wizard with the whole product appended to it.
`Meelmo-Community-Setup-<version>-…`	The free Community Edition, for Windows or Linux. It is the same wizard, with no licence page and no recorder.
`Meelmo-Community-Edge-<version>.ova` / `…-amd64.qcow2`	The Community Edge virtual machine. See The Community Edge VM.

Check a Linux download before you open it, and allow it to run. A browser saves every download without permission to execute.

```
sha256sum -c Meelmo-Setup-<version>-linux-x64.run.sha256
gpg --verify Meelmo-Setup-<version>-linux-x64.run.asc Meelmo-Setup-<version>-linux-x64.run
chmod +x Meelmo-Setup-<version>-linux-x64.run
```

On Windows, the elevation prompt must name the real publisher. *Unknown publisher* means the file was not signed. Return it to the vendor rather than clicking through.

Which role

One question decides everything else: **which process holds the node**. A node owns one plant's configuration, accounts and history, and exactly one program may hold it at a time.

Role	What holds the node	What starts with the machine
Workstation	The browser host, as a service, when the **Browser HMI** is installed. Without it, the Desktop holds the node.	The browser host's service, if installed. Nothing otherwise.
Plant server	The recorder service. Everything else on the machine is its client.	The recorder, and the browser host if installed.
Client of a Meelmo server elsewhere	Nothing on this machine. It is a client of a plant server.	Nothing.
Decide later	Nothing.	Nothing. The files are installed and no service is registered.

Choose **Plant server** when acquisition must run with nobody signed in and survive reboots. Choose **Workstation** for one person's machine that holds its own plant. Choose **Client** for an engineer's laptop or a control-room PC that watches a plant server. The *Configuration and deployment* chapter shows how to change the role later.

Network ports

Port	Program	Used for	Open to other machines
8443 TCP	Browser host	The browser HMI. On a workstation it is also the node's control plane.	When **Other machines on the plant network connect to this server** is ticked
8444 TCP	Recorder	The node's control plane: sign-in, accounts and configuration for the Desktop and the browser host.	Same

Port	Program	Used for	Open to other machines
9701 TCP	Recorder or browser host	The acquisition carrier: live values for attached clients.	Same
5353 UDP	Recorder or browser host	Node discovery (**Find nodes on this network**).	Same
7443 TCP	Recorder	The data-access API: history for the browser host, the SDKs, Grafana and MCP.	Only if you configure it
5077 TCP	Recorder	The read-only status API (`/health`, `/status`). Off by default.	Only if you configure it
8490 TCP	Meelmo Setup	The one-time setup page on a server with no screen, only while setup runs.	While setup runs
7445 TCP	Data-access server, run on its own	See the *Data access and SDKs* chapter. No installer registers it.	Only if you configure it

The Meelmo Setup wizard

Meelmo Setup is a window. On a server without a display it is served as a page in a browser instead (see A server with no screen); the pages are the same. The left rail shows the program name, a **Language** list and the **Steps** of this run, with the current step highlighted. The buttons are **Back**, **Next** (which becomes **Install**, **Repair** or **Remove** on the summary, and **Close** at the end) and **Cancel**.

Setup speaks the ten languages the product ships: English, German, Spanish, French, Friulian, Italian, Japanese, Korean, Russian and Chinese. It starts in the language of your desktop. On Linux this is read from `LANGUAGE`, `LC_ALL`, `LC_MESSAGES` and `LANG`, in that order. `--language de` (or `fr`, `ja` and so on) starts it in another language. Changing the **Language** list on any page keeps everything you have typed. The installation log stays in English, because it is what support reads.

The steps shown depend on your answers. A plant server is never asked **Server**, a client is never asked **Network**, and the Community Edition has no **Licence** step.

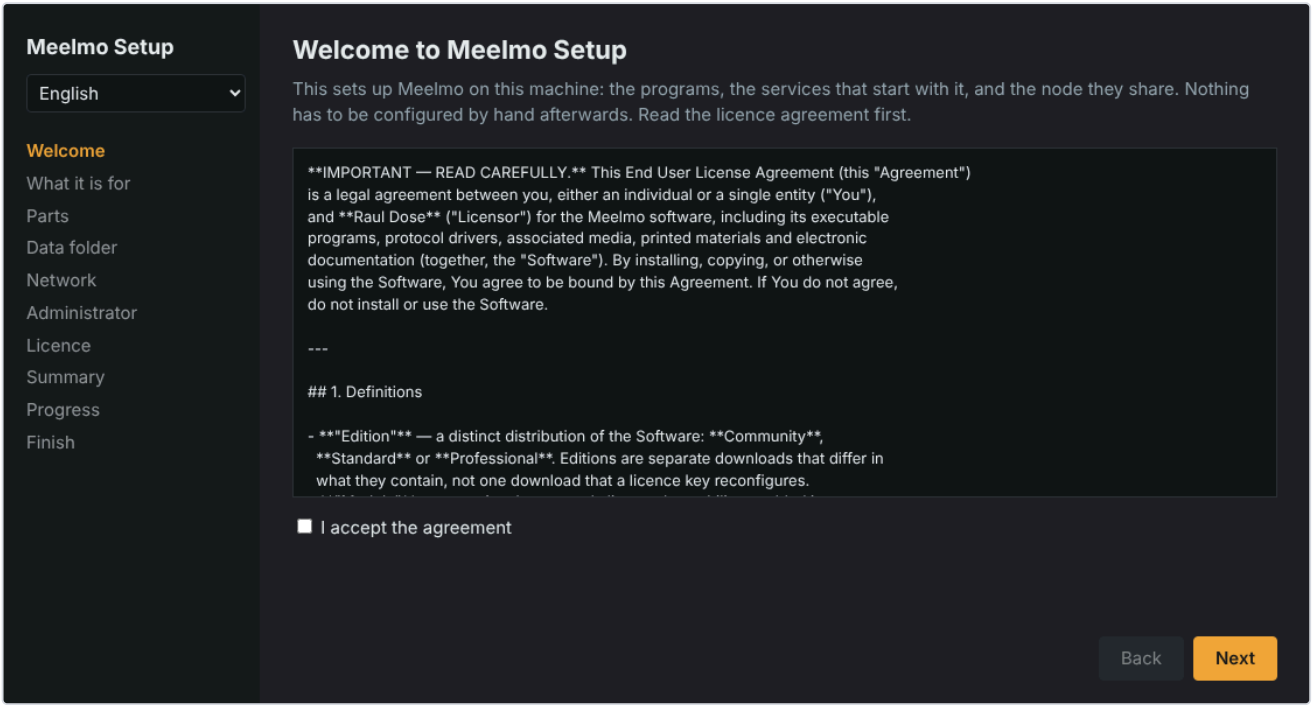
Step	Page title	Shown when
Welcome	*Welcome to Meelmo Setup*	Nothing is installed yet
Meelmo on this machine	*Meelmo is installed on this machine*	Meelmo is already installed (see Repair, change, upgrade and remove)
What it is for	*What will this machine do?*	Installing or changing
Parts	*What should be installed?*	Installing or changing
Data folder	*Where should this machine keep its data?*	Installing or changing
Server	*Which server does this machine use?*	Role is **Client of a Meelmo server elsewhere**
Network	*Who reaches this server?*	Role is **Plant server**
Administrator	*Who administers this node?*	Role is **Plant server**, or **Workstation** with the **Browser HMI**
Licence	*Licence*	Every edition except Community

Step	Page title	Shown when
Summary	*Ready to install*, *Ready to repair Meelmo* or *Ready to remove Meelmo*	Always
Progress	*Installing Meelmo* or *Removing Meelmo*	Always
Finish	*Finished*	Always

If an answer cannot be used, **Next** does not move on. The page shows why, in red, while you can still change it. The full list of these messages is in What setup refuses, and why.

Welcome

The licence agreement, and **I accept the agreement**. You cannot go on without ticking it. The Community Edition's page reads *Welcome to Meelmo Setup — Community Edition*. It says the edition needs no licence and does not expire.



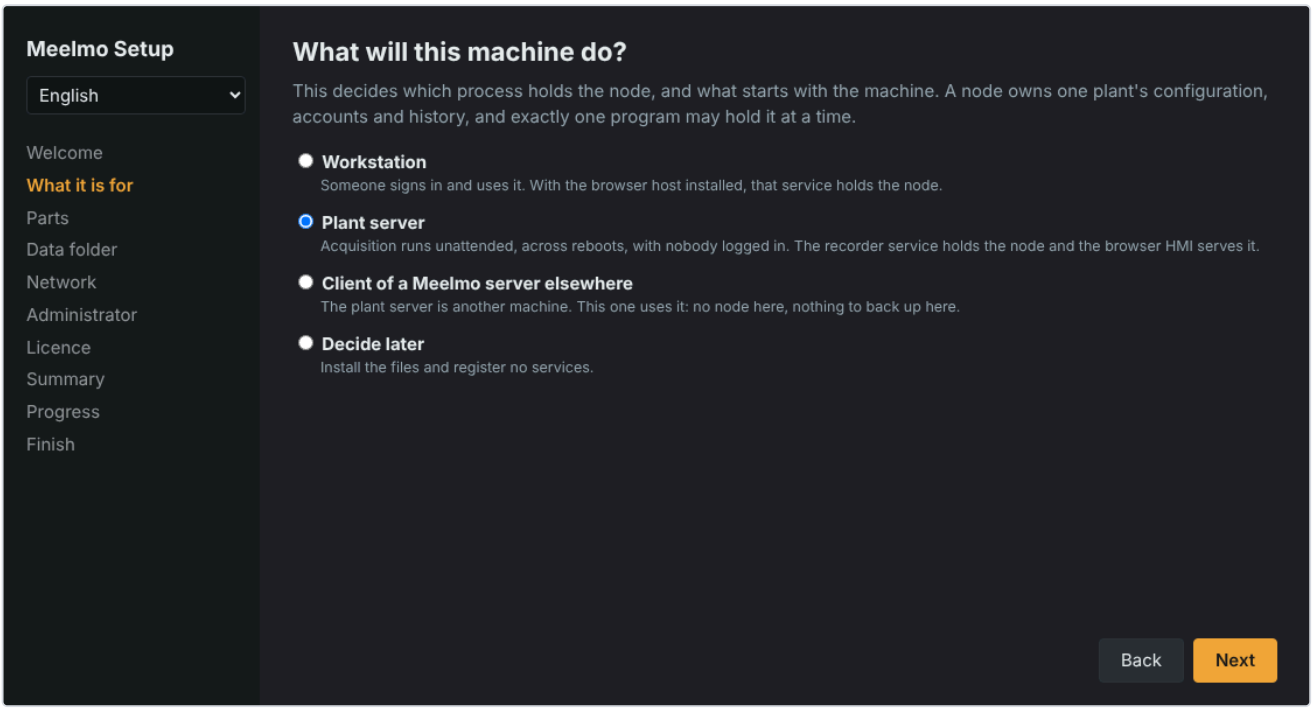
Meelmo Setup's Welcome page, here served to a browser: the licence agreement in a scrolling box, I accept the agreement unticked beneath it, and the Steps of this run down the left rail.

What will this machine do?

Four choices, each with a line of explanation:

Choice	Its description
Workstation	Someone signs in and uses it. With the browser host installed, that service holds the node.
Plant server	Acquisition runs unattended, across reboots, with nobody logged in. The recorder service holds the node and the browser HMI serves it.
Client of a Meelmo server elsewhere	The plant server is another machine. This one uses it: no node here, nothing to back up here.
Decide later	Install the files and register no services.

On a machine that already has Meelmo, the role it was given is selected. Pressing **Next** through an upgrade therefore changes nothing. **Plant server** cannot be chosen from a setup that does not carry the recorder, such as the Community Edition.

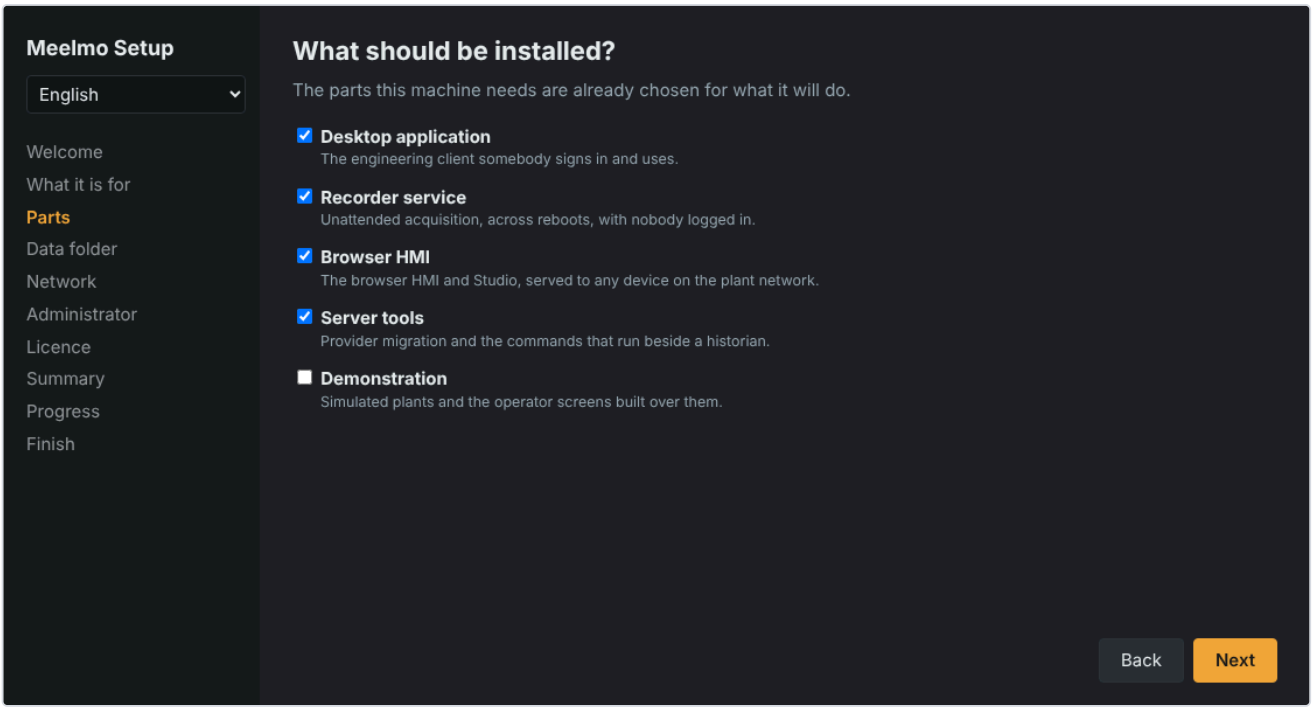


What will this machine do? on a server with no display: Workstation, Plant server (selected), Client of a Meelmo server elsewhere and Decide later, each with its line of explanation.

What should be installed?

The parts this setup carries, each with a tick box. Choosing a role ticks the parts that role needs, until you change the ticks yourself.

Part	Description	Ticked by default on Linux for
Desktop application	The engineering client somebody signs in and uses.	Workstation, Plant server, Client
Recorder service	Unattended acquisition, across reboots, with nobody logged in.	Plant server
Browser HMI	The browser HMI and Studio, served to any device on the plant network.	Workstation, Plant server, Client
Data-access server	The REST and gRPC surface integrations read through. It is installed as files; no role registers it as a service.	—
Server tools	Provider migration and the commands that run beside a historian.	Plant server
Demonstration	Simulated plants and the operator screens built over them.	Workstation
Database history providers	SQL Server, Oracle, TimescaleDB, QuestDB and ClickHouse. Listed only by a build where they are a separate part.	—



What should be installed? on Linux for a plant server: Desktop application, Recorder service, Browser HMI and Server tools ticked, Demonstration not.

On Windows every part the setup carries is ticked whatever the role. The role decides what is *registered*; a part nothing uses is files on a disk. **Desktop application** cannot be unticked on Windows, because the program folder is the Desktop's folder. **Decide later** on Linux ticks everything the setup carries.

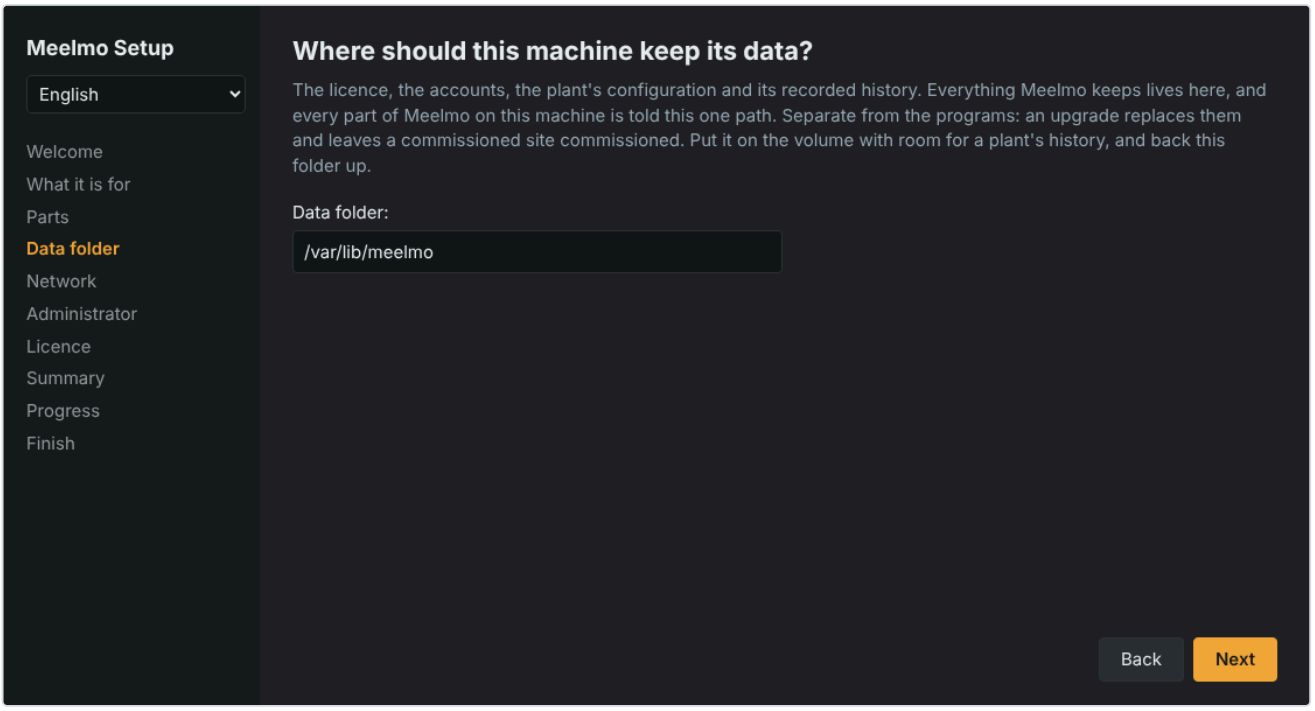
Where should this machine keep its data?

Data folder: is the one folder that holds everything Meelmo keeps:

- the licence and its entitlement file;
- the node database, its accounts and its audit trail;
- the sealed secrets and their master key;
- the site's certificate authority;
- the recorder's configuration;
- the recorded history and the logs.

Every Meelmo program on the machine is told this one path. The folder is kept separate from the programs, so an upgrade replaces the programs and leaves a commissioned site commissioned. Put it on the volume with room for a plant's history, and back it up.

Platform	Offered	Rules
Linux	`/var/lib/meelmo`	A full path starting with `/`. Not inside `/opt/meelmo`, `/opt/meelmo-recorder`, `/opt/meelmo-web`, `/opt/meelmo-tools`, `/opt/meelmo-setup` or `/usr`, which are replaced on every upgrade.
Windows	`C:\Meelmo\Data`	A drive and folder (`D:\Meelmo\Data`) or a share (`\\server\meelmo`). Not inside the program folder or the Windows folder.



Where should this machine keep its data? on Linux, offering /var/lib/meelmo as the Data folder.

On a **new** Windows installation the same page also asks **Program folder:**, and offers `C:\Program Files\Meelmo`. An upgrade deletes everything in the program folder that is not Meelmo's, so the folder must be:

- a full path, and not a drive on its own;
- outside the Windows folder;
- outside the data folder;
- empty, new, or already a Meelmo installation.

An installed machine keeps the program folder it has.

Which server does this machine use?

Shown for **Client of a Meelmo server elsewhere**.

Field	What to enter
Server:	The name or address of the plant server. Leave it **blank** to have this machine ask the network: nodes announce themselves, and when exactly one answers, this installation is pointed at it.
Administrator:	An administrator of that server's node. It is the account created when that machine was installed. Required: this machine asks the node for a credential of its own, and nothing on the network can answer that for you.
Password:	That administrator's password.

Who reaches this server?

Shown for **Plant server**. It has one tick box: **Other machines on the plant network connect to this server**.

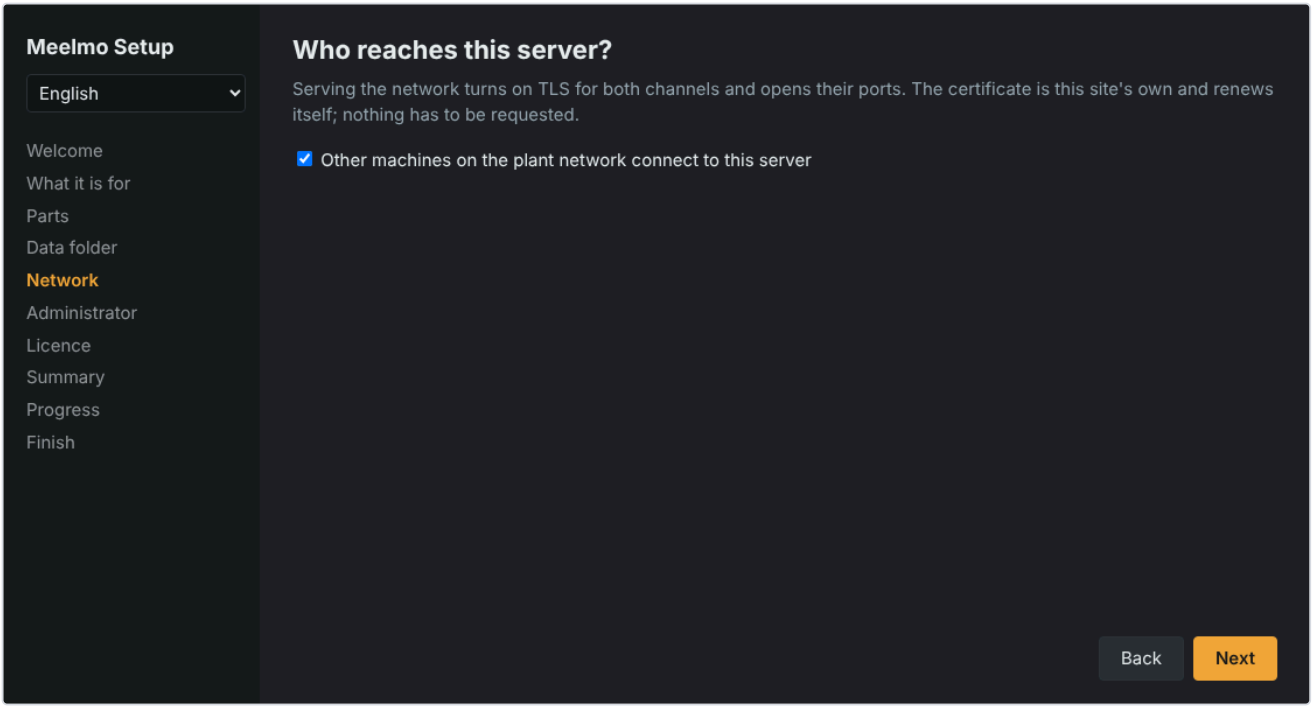
Ticked, both channels (the control plane and the acquisition carrier) bind every interface over TLS. The firewall is opened for:

- 8444 TCP, the node control plane;

- 9701 TCP, acquisition;
- 5353 UDP, discovery;
- 8443 TCP, the browser HMI, when it is installed.

The certificate is the site's own. The recorder creates the site's certificate authority on its first start, issues itself a certificate and renews it. Nothing has to be requested.

Unticked, everything listens on this machine only. When setup runs in a browser on a server with no screen, the box starts ticked, because the browser is on another machine.

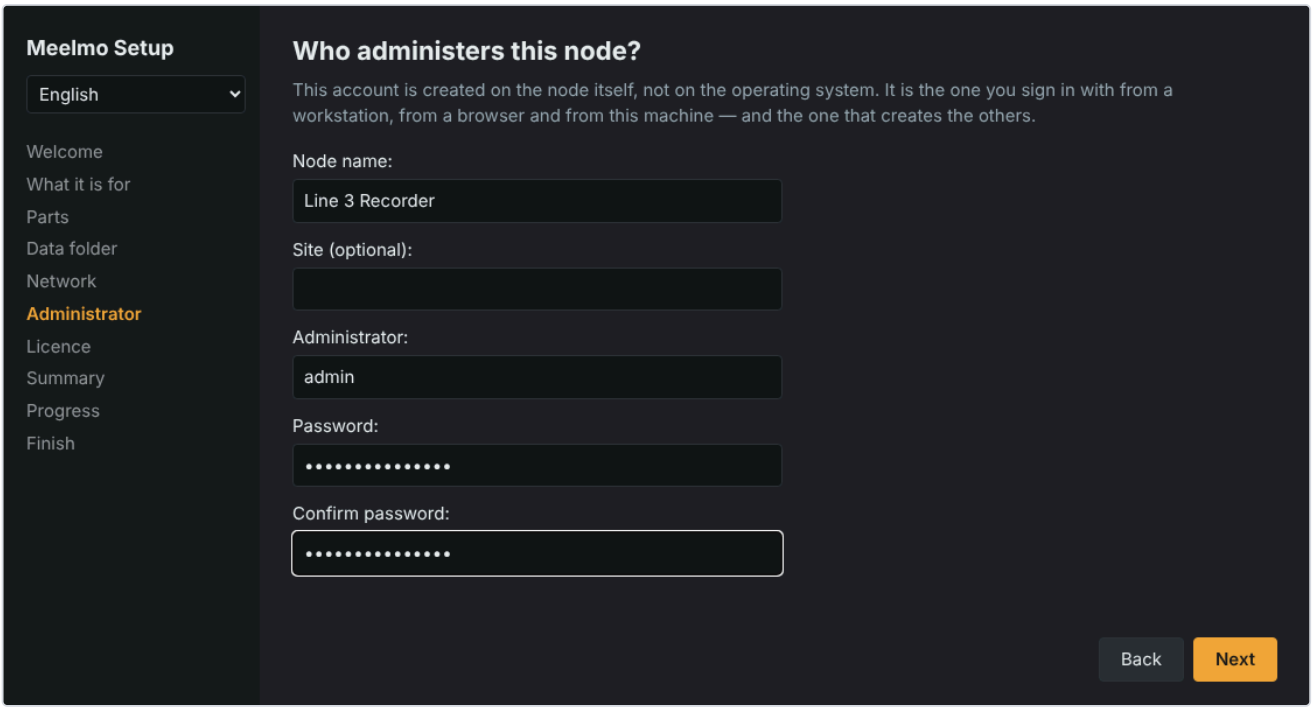


Who reaches this server? in the browser version of the wizard, with Other machines on the plant network connect to this server already ticked.

Who administers this node?

Shown when this machine commissions a node: on a **Plant server**, and on a **Workstation** with the **Browser HMI**.

Field	Default	Meaning
Node name:	This machine's name	What a workstation sees when it finds this server. Required.
Site (optional):	Blank	The site's name.
Administrator:	`admin`	The first administrator, created **on the node**, not in the operating system. It is the account you sign in with from a workstation, from a browser and from this machine. It creates every other account.
Password:, **Confirm password:**	—	Held to the node's password policy: 8 to 256 characters, with at least three of lowercase, uppercase, digit and symbol.
Demonstration to put into service:	**None — this is a real plant**	Offered only when the **Desktop application** and **Demonstration** parts are ticked and the node is new. The choices are **Packaging line (with its 3D model)**, **Galvanizing line**, **Meltshop**, **Hot strip mill** and **Reference works**.



Who administers this node? with Node name Line 3 Recorder, Site left empty, Administrator admin and the password typed twice. Demonstration is not offered, because that part was not ticked.

A node that already has an administrator. Uninstalling never removes the data folder, so a reinstall usually finds the node the previous installation made. The page then says so and asks for **one of that node's existing administrators**. Setup signs in with it to finish setting the node up. It creates no account and changes no password. The demonstration is not offered.

A demonstration, if chosen, is the installation's last step. It is done as the administrator you named:

- the plant becomes the node's active configuration and starts acquiring from its simulator;
- its saved queries and reports are written to the node;
- with the browser host installed, its operator screens are imported and published. The packaging line's 3D model is put in the asset library.

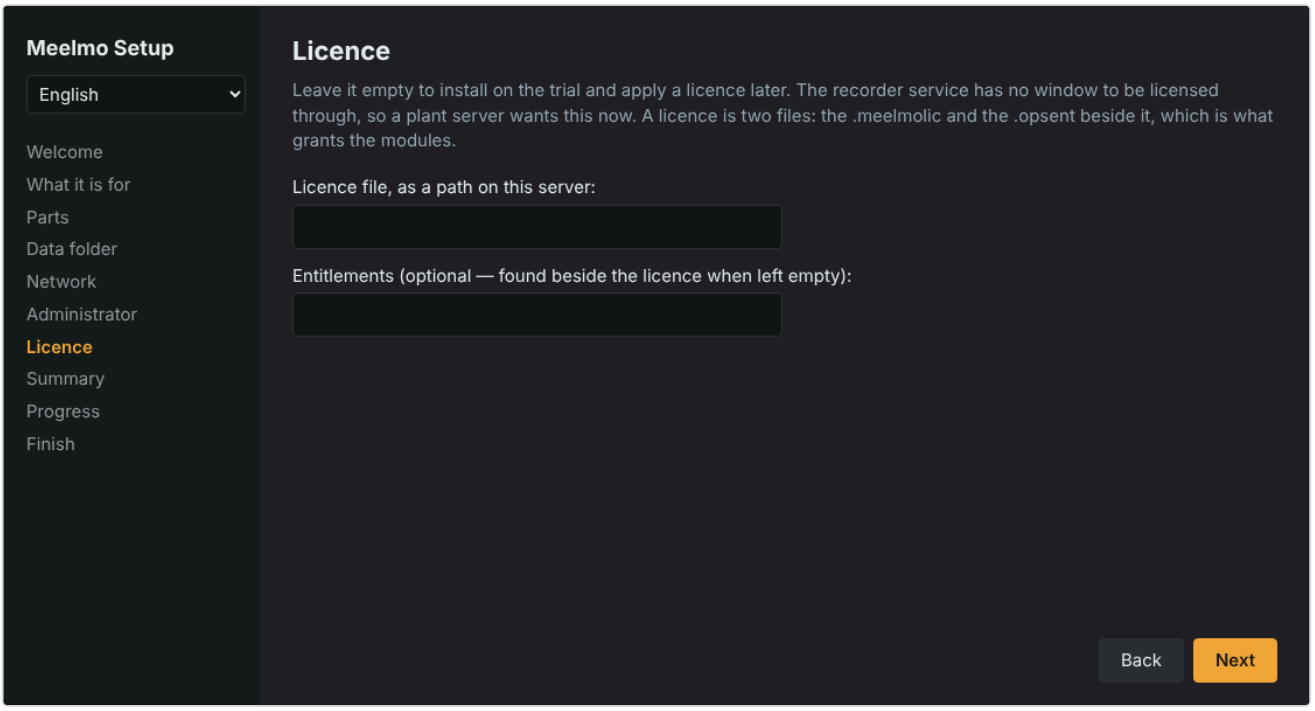
A step that cannot be done is written in the log and fails nothing. **Help ▸ Demonstration…** in the Desktop loads a plant at any time.

Licence

Not shown by the Community Edition.

Field	Meaning
Licence file:	The `.meelmolic` file, with **Browse…**.
Entitlements (optional — found beside the licence when left empty):	The `.opsent` file that grants the modules.

Leave both empty to install on the trial and apply a licence later. **A plant server wants the licence now**: the recorder service has no window to be licensed through. In the browser version of the wizard, the first field reads **Licence file, as a path on this server:**. Give the path on the server, not on the machine your browser runs on.



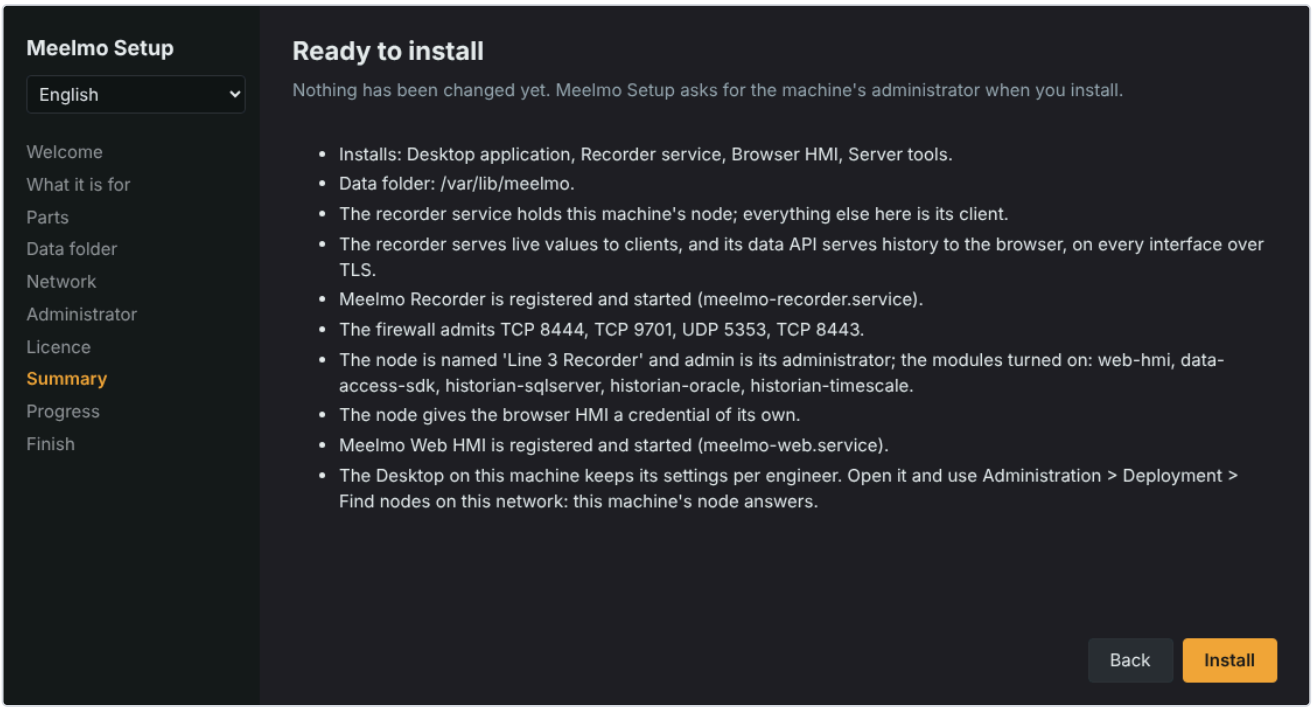
The Licence page in a browser: Licence file, as a path on this server, and Entitlements, both left empty to install on the trial.

Summary

Nothing has been changed yet. Then, in plain sentences, what will happen:

- the parts installed and the data folder;
- which process holds the node;
- each service that is registered and started;
- which ports the firewall admits;
- the node's name and its administrator;
- each credential the node gives out;
- any note, such as *The Desktop on this machine keeps its settings per engineer…*.

The password is never shown. The Community Edition adds a line with the edition's limits.



Ready to install for a plant server: the parts and data folder, the recorder service holding the node, the services registered and started, the ports the firewall admits, the node's name and administrator, and the note about the Desktop. The button is now Install.

Press **Install**. Meelmo Setup now asks for the machine's administrator: polkit on Linux, which asks in its own dialog, and the elevation prompt on Windows. Meelmo Setup never sees that password.

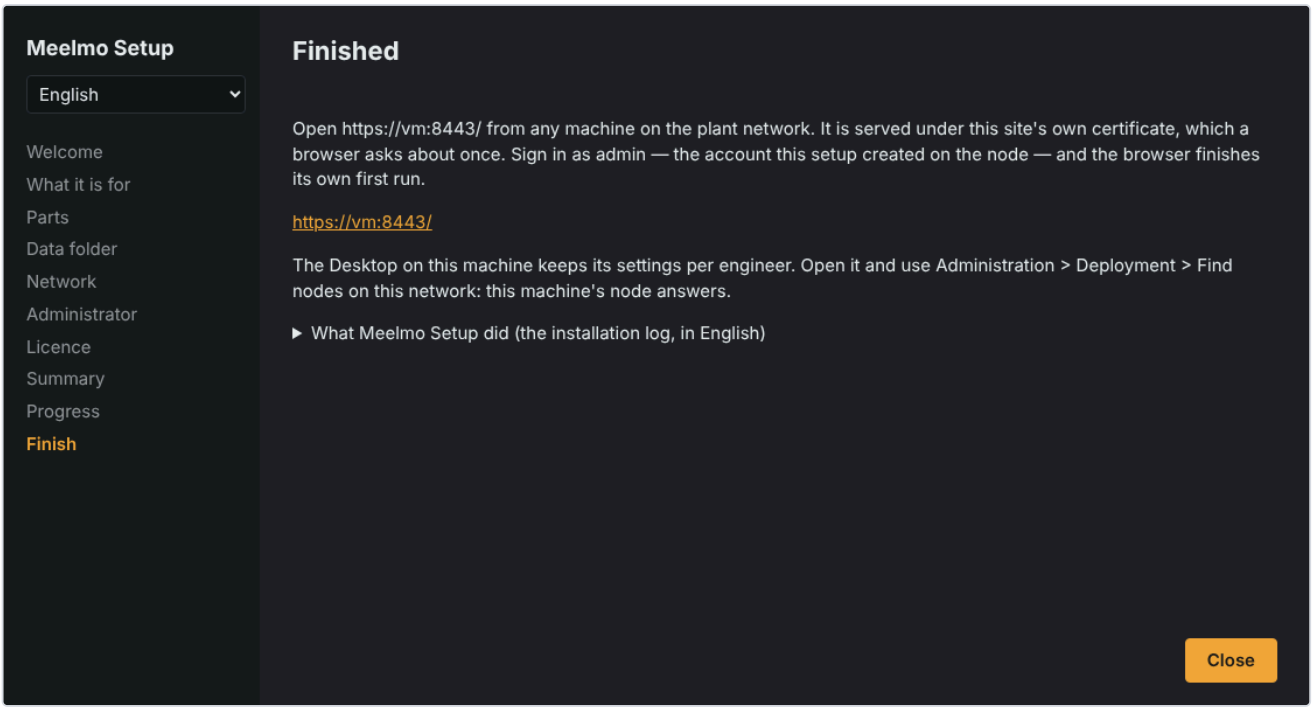
Installing Meelmo

This takes a minute: the node is started, named and given its administrator. The page shows each step as it happens.

Finished

What the page shows depends on how the run ended:

Outcome	What you see
Succeeded, with a browser host	*Open https://pkg-srv-01:8443/ from any machine on the plant network...* (plant server serving the network) or *Open http://127.0.0.1:8443/ in a browser on this machine...*, then *Sign in as admin*. An **Open ...** button opens it.
Succeeded, no browser host	*Start Meelmo from the applications menu.*
The node was commissioned	**Read-only credential for integrations (MCP, the SDKs, Grafana).** A box with the token. Copy it now: the node stores it hashed and cannot show it again.
The node refused something	One **Refused:** line per refusal, in the node's words.
Something was left for later	The notes, for example *No administrator was given, so the node was not commissioned here. The browser's first-run page commissions it.*
Did not finish	*Meelmo Setup did not finish. What it did and where it stopped is above, and in the log in the data folder's logs directory.*



Finished on a plant server serving the network: open https://vm:8443/ (this server's own name) from any machine and sign in as admin, the link to it, the note about the Desktop, and What Meelmo Setup did, folded.

What Meelmo Setup did (the installation log, in English) expands to the full log. The same log is kept under the data folder's `logs`.

What setup refuses, and why

Message	Page	What to do
Accept the licence agreement to install Meelmo.	Welcome	Tick **I accept the agreement**.
This setup does not carry the recorder, and a plant server is the machine the recorder runs on.	Role	Use the full edition's setup, or choose another role.
Choose at least one part to install.	Parts	Tick a part.
A plant server needs the recorder service: it is what holds the node and acquires unattended.	Parts	Tick **Recorder service**.
Give the data folder as a full path, such as ...	Data folder	Give an absolute path.
... is inside ..., which is replaced on every upgrade. Keep the plant's data outside it.	Data folder	Move the data folder out of the program folder.
Give the program folder as a full path, such as C:\Program Files\Meelmo — not a drive on its own.	Data folder (Windows)	Name a folder.
... is inside the data folder .... Keep the programs and the plant's data apart.	Data folder (Windows)	Choose another program folder.
... already holds other files, and everything in the program folder that is not Meelmo's is removed on the next upgrade. Choose an empty folder or a new one.	Data folder (Windows)	Choose an empty or new folder.
Name an administrator of that server's node...	Server	Enter an administrator of the plant server.
Give this node a name...	Administrator	Enter **Node name:**.

Message	Page	What to do
Name the first administrator. Somebody has to be able to sign in.	Administrator	Enter **Administrator:**.
The two passwords do not match.	Administrator	Type the same password twice.
Set a password for the administrator.	Administrator	Enter a password.
Password must be at least 8 characters. / …at most 256 characters. / …must include at least 3 of: lowercase, uppercase, digit, symbol.	Administrator	Choose a password that meets the policy.
There is no file at ….	Licence	Correct the path, or leave the field empty for the trial.

Worked example: the packaging line's plant server, on Linux

A new Linux machine, `pkg-srv-01`, is to acquire the packaging line unattended and serve the browser HMI to the control room.

1. Copy `Meelmo-Setup-<version>-linux-x64.run` to the machine, check it and allow it to run (see Which file). Open it from the file manager.
2. **Welcome**: read the agreement, tick **I accept the agreement**, **Next**.
3. **What will this machine do?**: **Plant server**, **Next**.
4. **What should be installed?**: leave **Desktop application**, **Recorder service**, **Browser HMI** and **Server tools** ticked. Tick **Demonstration** as well, so the machine finishes acquiring a plant. **Next**.
5. **Where should this machine keep its data?**: on this machine the history volume is mounted at `/srv/meelmo`, so type `/srv/meelmo` into **Data folder:**. **Next**.
6. **Who reaches this server?**: tick **Other machines on the plant network connect to this server**. **Next**.
7. **Who administers this node?**:
   - **Node name:** `PKG-SRV-01`;
   - **Site (optional):** `Plant North`;
   - **Administrator:** `commissioner`, with a password such as `Pallet-Wrap-2026` typed twice;
   - **Demonstration to put into service: Packaging line (with its 3D model)**.

   Next.
8. **Licence**: **Browse…** to `site.meelmolic`. Leave **Entitlements** empty: the `.opsent` beside it is found. **Next**.
9. **Summary** reads, among other lines:
   - *The recorder service holds this machine's node; everything else here is its client.*
   - *The firewall admits …*
   - *The node is named 'PKG-SRV-01' and commissioner is its administrator; …*

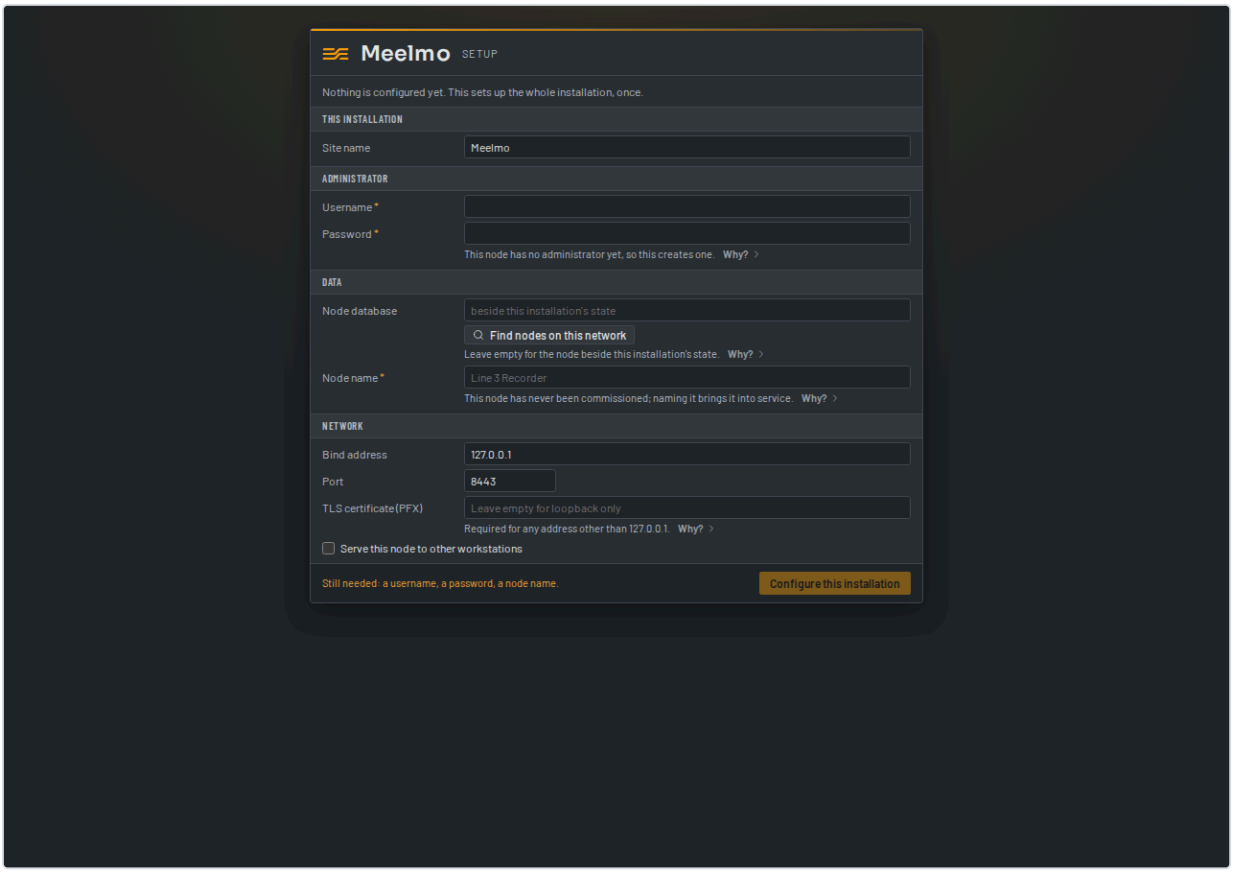
   Press **Install** and give the machine's administrator password to polkit.
10. **Finished**:
    - Copy the read-only credential into the site's password vault.
    - Press **Open** **https://pkg-srv-01:8443/**. Your browser asks once about the site's certificate.

- Sign in as `commissioner`. The packaging line's screens are already published.

The Desktop on this Linux machine keeps its settings per engineer. Open it and use **Administration ▸ Deployment… ▸ Find nodes on this network** to attach it to `PKG-SRV-01`.

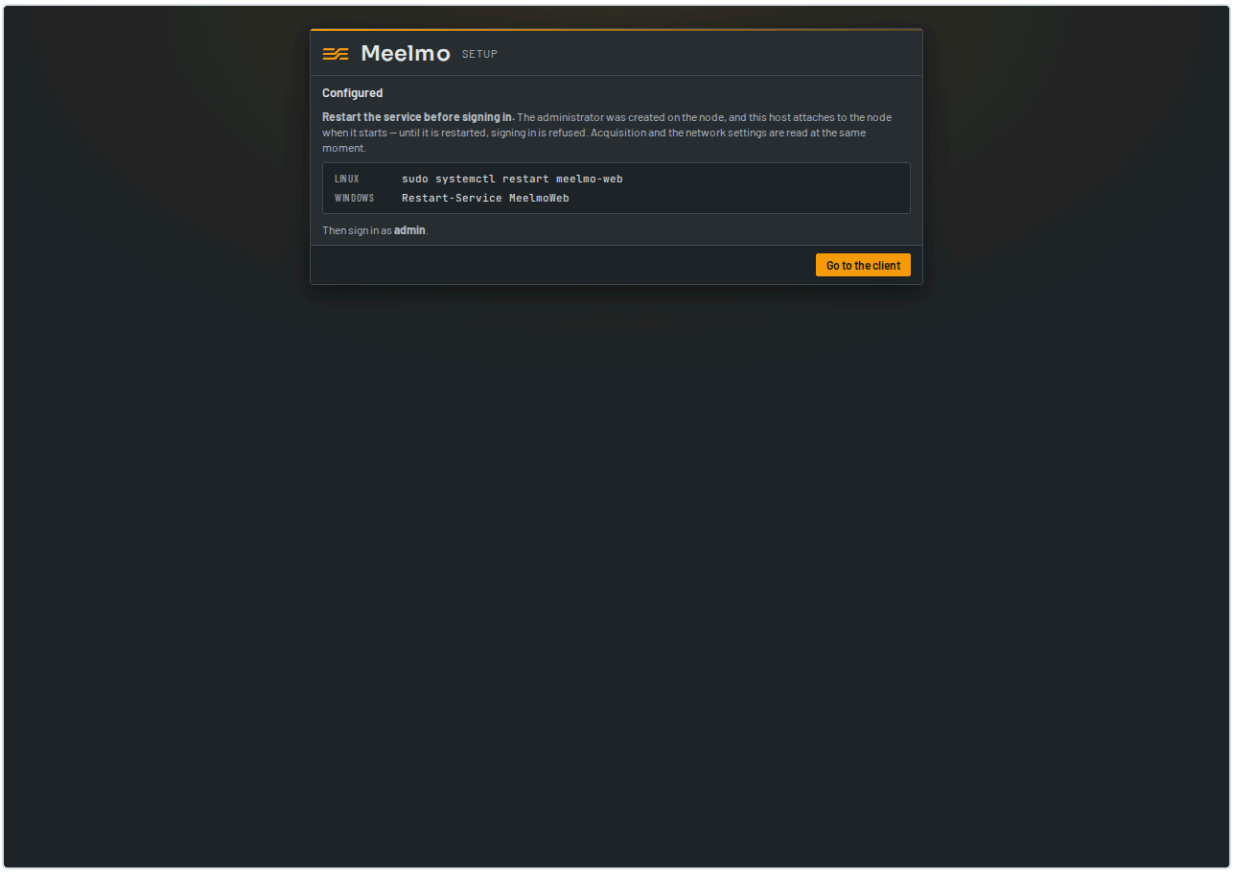
When no administrator was given

A node that setup did not commission waits for the first person to sign in. On a workstation with the browser host, that person opens `http://127.0.0.1:8443/`. They are taken to the browser's first-run page. It asks for the site name, an administrator, the node's name and the network settings. The *Web administration* chapter describes it field by field, where your copy of this manual includes it.



The browser's first-run setup page on a new installation: site name, administrator username and password, node database with Find nodes on this network, node name, bind address 127.0.0.1, port 8443, an empty TLS certificate and Serve this node to other workstations. The line at the bottom says what is still needed.

Configure this installation creates the administrator on the node. If the page then asks you to restart the service, do so before signing in.



The first-run page after Configure this installation: Configured, Restart the service before signing in, with the Linux and Windows commands to restart the browser host, then Go to the client.

On a plant server, the first-run page also asks for the node's **one-time claim**. The recorder writes it in its log when it starts:

```
journalctl -u meelmo-recorder -b | grep -i -A2 "one-time claim"
```

On a workstation where the Desktop holds the node (no browser host), the Desktop's own first-run screen asks for the node name and first administrator. See the *Getting started* chapter.

A server with no screen

A plant server usually has no display. When Meelmo Setup finds none (`DISPLAY` and `WAYLAND_DISPLAY` unset), or when you start it with `--headless`, it serves the same wizard as a page in a browser instead of a window.

Start it as root over SSH. There is no desktop on the server to ask for the administrator:

```
sudo ./Meelmo-Setup-<version>-linux-x64.run
```

Started as another user, it asks polkit (`pkexec`) for the administrator. If polkit is not there, it says *Meelmo Setup installs services, so it needs the machine's administrator, and polkit is not here to ask for it. Start it again as root.* It then prints:

```
Meelmo Setup is waiting for you in a browser. Open:
    https://pkg-srv-01:8490/?key=3f9c…
The page's certificate was made for this run, so your browser will ask before it trusts it.
```

```
Its SHA-256 fingerprint is 4F:1A:….
This page answers only that address and key, and only until setup has finished.
```

```
$ ./meelmo-setup --headless --listen 127.0.0.1 --port 8491 --payload payload --destdir root
Meelmo Setup is waiting for you in a browser. Open:
    https://127.0.0.1:8491/?key=43ebd445bc1efdf149fd1d356c179d53

The page's certificate was made for this run, so your browser will ask before it trusts it.
Its SHA-256 fingerprint is 57:70:66:9F:4C:9E:89:49:08:5A:33:44:6C:3B:40:86:6B:C5:50:A4:D0:FD:79:2C:65:2B:39:5C:80:05:A
8:75.
This page answers only that address and key, and only until setup has finished.
```

Meelmo Setup started with --headless: the address with its one-time key, the certificate's SHA-256 fingerprint to compare in the browser, and how long the page answers.

To finish the installation from your browser:

1. Open the address from any machine.
2. Compare the certificate fingerprint your browser shows with the printed one, and continue only if they match.
3. Answer the pages.

The page speaks your browser's language until you pick another. A request without the key is refused (*This page needs the key Meelmo Setup printed with its address*). The page is served only over TLS, because the administrator's password is typed into it.

The page stops when you close the finish page, or a quarter of an hour after setup has finished. Its exit code is 0 when setup succeeded and 1 otherwise. On a headless server the role starts at **Plant server** (when the recorder is carried), and **Other machines on the plant network connect to this server** starts ticked.

Option	Default	Meaning
`--headless`	Only when there is no display	Serve the page even where a window could be drawn.
`--listen ADDRESS`	`0.0.0.0`	The address the page listens on.
`--port N`	`8490`	The port.
`--key KEY`	Random for each run	The one-time key the address carries.
`--language CODE`	The terminal's language	The language of the console lines and of the page, until a choice is made on the page.

Answer files and unattended installation

Every question the wizard asks can be answered in advance in an answer file. This is how a site images many machines without somebody at each keyboard. It is an INI file with one section, `[Meelmo]`. A file with no section header is read as `[Meelmo]`. Lines starting with `;` or `#` are comments.

Key	Values	Answers
`Role`	`workstation`, `plantserver`, `client`, `later`	**What will this machine do?**

Key	Values	Answers
`Components`	`main` (or `desktop`), `recorder`, `web`, `server`, `tools`, `demo`, `providers`, separated by commas or spaces	**What should be installed?** Omitted means everything the setup carries.
`DataDir`	A full path	**Data folder:**
`ProgramDir`	A full path (Windows only)	**Program folder:**
`NodeName`	Text	**Node name:**
`Site`	Text	**Site (optional):**
`Administrator`	An account name	On a plant server or workstation, the administrator to create (or the existing one). On a client, an administrator of the server's node.
`Password`	Its password	Everything after the first `=`. It may contain `=`, `;` and `#`.
`ServeOtherMachines`	`yes` / `no` (also `true`, `1`, `on`)	**Other machines on the plant network connect to this server**
`Server`	A name or address, or blank	**Server:** on a client. Blank asks the network.
`Licence` (or `License`)	Path to the `.meelmolic`	**Licence file:**
`Entitlements`	Path to the `.opsent`	**Entitlements**
`Demonstration`	`packaging`, `galvanizing`, `meltshop`, `hotstrip`, `reference`, `none`	**Demonstration to put into service:**
`DesktopIcon`	`yes` / `no` (Windows only; default yes)	A desktop shortcut as well as the Start menu entry.
`ReplaceAuthority`	A SHA-256 fingerprint	No page asks this. It accepts a site root certificate that replaces the one this machine already trusts, for a reinstall or a plant that re-created its authority. Compare the fingerprint with the plant server's own before you add it.

Setup does not guess:

- **An unknown role is not guessed at.** `Role=plant-server` installs the files and registers nothing, and says *Role=plant-server is not one this installer knows (workstation, plantserver, client, later), so nothing was registered.*
- **An unknown key or component is reported.** For example, *site.ini: 'Nodename2' is not a key this installer knows, so it was ignored.*
- **A missing file stops the run.** An answer file named on the command line that is not there stops the installation before anything is done. *Somebody asked and the answer was lost* must not look like *nobody asked*.
- **No answers at all means *Decide later*.** A silent installation given no answers installs the files and registers no services.
- **On Windows, a weak password stops the run.** The Windows installer checks the password in the file against the node's policy before a file is copied. This applies to the workstation and plant server roles.

The file carries the administrator's password, which is why it is a file and not a command line: a command line is visible in the process list for as long as setup runs. Protect the file, and delete it afterwards.

Platform	Command
Linux	`sudo ./Meelmo-Setup-<version>-linux-x64.run --unattended --answers site.ini`
Windows, Meelmo Setup	`Meelmo-Setup-<version>-win-x64.exe --unattended --answers site.ini`
Windows installer	`Meelmo-<version>-setup.exe /VERYSILENT /ANSWERS=C:\deploy\site.ini` (add `/LANG=german`, `spanish`, `french`, `italian`, `japanese`, `korean` or `russian` for the installer's own messages)

Worked example: fifty control-room clients

Every PC in the packaging hall's control room is to be a client of `pkg-srv-01`. A deployment tool copies this file to each PC as `C:\deploy\client.ini`:

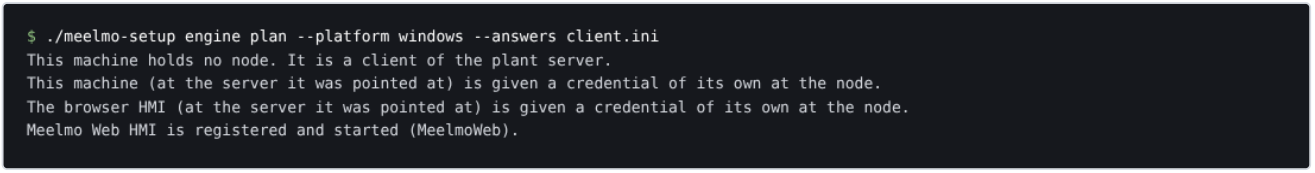
```
[Meelmo]
Role=client
Server=pkg-srv-01
Administrator=commissioner
Password=Pallet-Wrap-2026
Components=main,web
DesktopIcon=no
```

It then runs:

```
Meelmo-Setup-<version>-win-x64.exe --unattended --answers C:\deploy\client.ini
```

and deletes the file. Each PC asks `PKG-SRV-01` for a credential of its own, using `commissioner` once. The password is not kept on the PC.

Before rolling out, run `plan` on one PC (see The setup command line). It prints what the file would do without changing anything.



engine plan for client.ini: this machine holds no node and is a client of the plant server, it and its browser HMI are each given a credential of their own at the node, and Meelmo Web HMI is registered and started (MeelmoWeb).

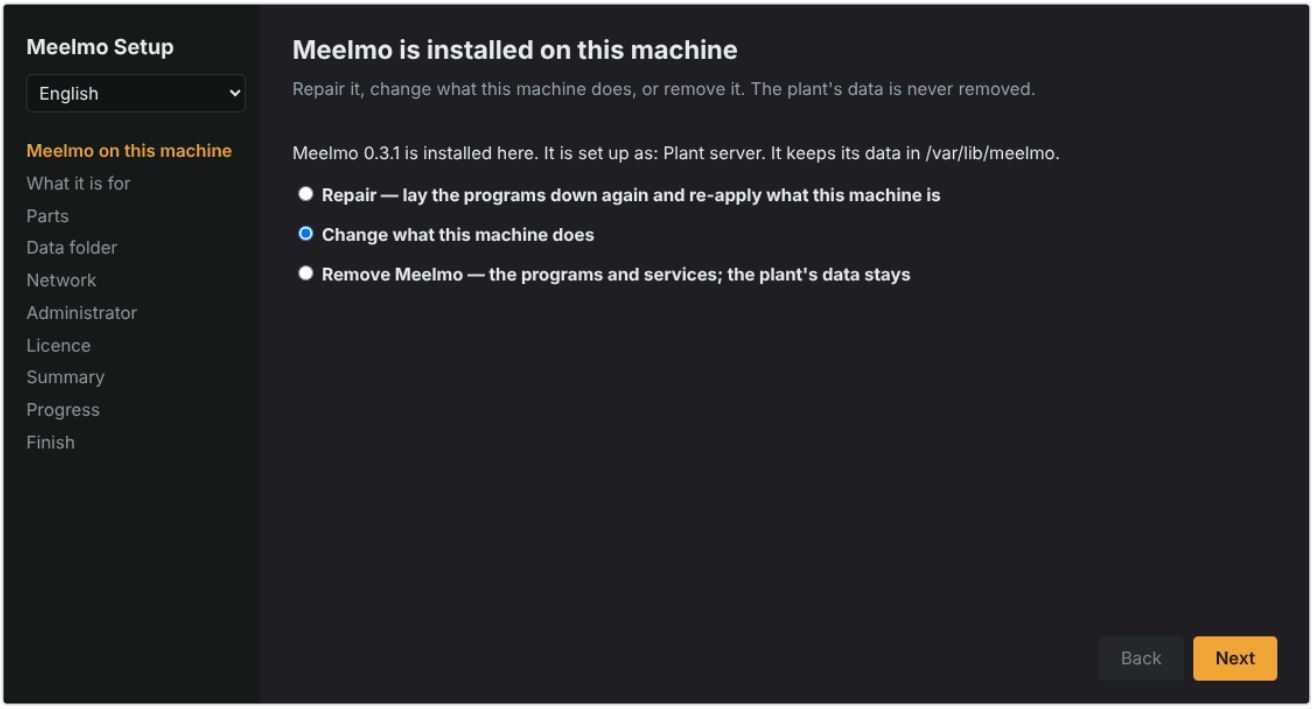
Repair, change, upgrade and remove

Open Meelmo Setup on a machine that already has Meelmo. It opens on *Meelmo is installed on this machine*, and says:

- *Meelmo <version> is installed here.*
- *It is set up as: <role>.*
- *It keeps its data in <folder>.*

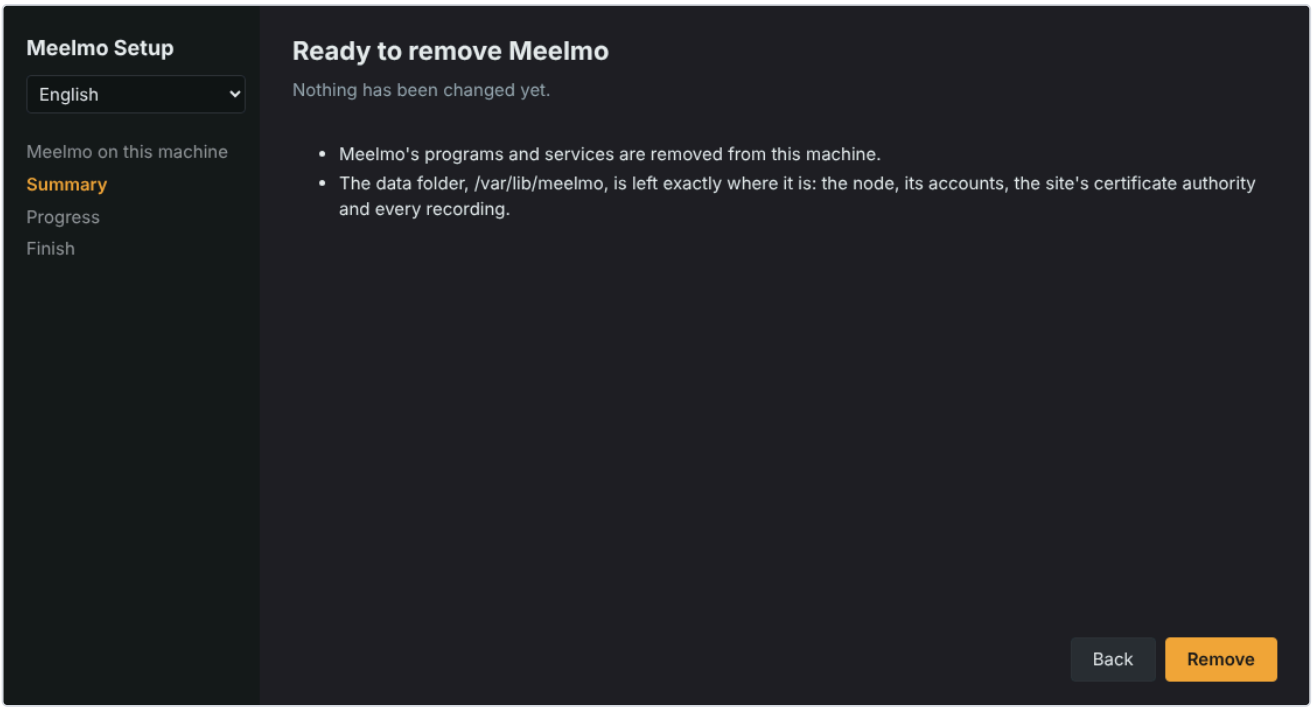
Then choose one of:

Choice	What it does
Repair — lay the programs down again and re-apply what this machine is	Goes straight to the summary, then reinstalls the programs and re-applies the recorded role, data folder and parts. It asks for no password and loads no demonstration.
Change what this machine does	The ordinary pages, opened on the machine's current answers. This is also how you **upgrade**: open the newer Meelmo Setup, choose this, and press **Next** through it.
Remove Meelmo — the programs and services; the plant's data stays	Removes the programs, the services and Meelmo Setup's own copy.



Meelmo is installed on this machine: Meelmo 0.3.1, set up as Plant server, keeping its data in /var/lib/meelmo, with Repair, Change what this machine does (selected) and Remove Meelmo.

The data folder is never removed. It holds the licence, the accounts, the node and its audit trail, the site's certificate authority and every recording. Delete it by hand only if you mean all of that.



Ready to remove Meelmo: the programs and services are removed, and the data folder, /var/lib/meelmo, is left exactly where it is. The button is now Remove.

An upgrade:

- replaces the programs;
- keeps the data folder exactly as it is;
- keeps an existing recorder configuration;
- never overwrites a licence;
- stops each service before its program is replaced, and starts it again afterwards.

On a machine that holds a node, it asks for one of that node's administrators (see Who administers this node?).

Two editions. The Community Edition's setup, opened on a machine with the paid edition, offers only **Remove**. It says *This machine has the paid edition of Meelmo. This is the Community Edition's setup: it can remove Meelmo from this machine, and it does not replace the paid edition with the free one.* An answer file run unattended is refused the same way, before anything changes. The other way round is an ordinary upgrade: the paid setup installs over a Community machine and keeps its data.

The copy kept for maintenance. On Linux, Meelmo Setup keeps a copy of itself, without the product, in `/opt/meelmo-setup`, with a **Meelmo Setup** entry in the applications menu. That copy can only remove Meelmo. It says *This copy of Meelmo Setup was kept to remove Meelmo. To repair it or change what this machine does, open the Meelmo Setup you downloaded.*

Removing from the command line:

Platform	Command
Linux	`sudo /opt/meelmo-setup/meelmo-setup engine uninstall --platform linux`
Windows, Meelmo Setup	Programs and Features ▸ Uninstall opens the wizard on *Ready to remove Meelmo*. Silently: `--uninstall --unattended`.

The setup command line

The engine that does the work behind every page can be run on its own. It is useful for:

- checking an answer file before it is used;
- reading what a machine already has;
- removing Meelmo from a script.

Where it is:

- On Linux, and with Meelmo Setup for Windows, give the setup file `engine` as its first argument. For example, `sudo /opt/meelmo-setup/meelmo-setup engine status`.
- On a machine installed by `Meelmo-<version>-setup.exe`, it is `setup\Meelmo.Setup.exe` in the program folder.

Command	Options	What it does
`plan`	`--platform windows|linux --answers FILE [--state DIR] [--app DIR]`	Says what would be done, and does nothing.
`status`	`[--platform windows|linux] [--destdir ROOT]`	What this machine already has of Meelmo, as JSON.
`uninstall`	`[--platform windows|linux] [--destdir ROOT]`	Removes the programs and services. The data folder is never removed. Needs the machine's administrator.
`install`	`--payload DIR [--answers FILE] [--destdir ROOT] [--result -|FILE] [--platform windows|linux]`	Lays a payload down and applies the role. The wizard runs this for you.
`apply`	`--answers FILE --state DIR --app DIR [--token-out FILE] [--refusals-out FILE]`	Windows: applies the role to files the Windows installer has already put down. The installer runs this for you.
`pack`, `inspect`	`--program FILE …`	Release tooling for building a Meelmo Setup. Not used on a site.

`--destdir ROOT` stages everything into a directory standing in for the machine. It asks for no administrator and starts nothing. Use it to rehearse an installation.

Exit code	Meaning
`0`	Done, or everything the product can still finish later
`1`	A join the node answered, and this machine could not keep the credential
`2`	Asked for something that cannot be done: a missing option, no administrator, the Community Edition over the paid one
`3`	A failure while laying files down

`status` prints one JSON object with these fields:

Field	Meaning
`installed`	Whether Meelmo is installed
`version`	The installed version

Field	Meaning
`components`	The parts installed, by their answer-file names (`main` , `recorder` , ...)
`role`	`workstation` , `plantserver` , `client` or `later`
`edition`	`full` or `community`
`dataDir`	The data folder
`programDir`	The program folder, where one is recorded
`installedBy`	`meelmo-setup` , or `inno` for a machine installed by `Meelmo-<version>-setup.exe`

```
$ ./meelmo-setup engine status --platform linux --destdir root
{"installed":true,"version":"0.3.1","components":["main","recorder","web","tools"],"role":"plantserver","edition":"ful
l","dataDir":"/var/lib/meelmo","programDir":null,"installedBy":"meelmo-setup"}
```

engine status on a plant server installed by Meelmo Setup: one JSON object with installed, version, components, role, edition, dataDir, programDir and installedBy.

All the options of the Meelmo Setup file

Option	Meaning
`engine …`	Runs the engine (table above) instead of the wizard.
`--headless` , `--listen` , `--port` , `--key`	A server with no screen.
`--language CODE`	The language to start in.
`--unattended --answers FILE`	Install with nobody at the machine.
`--uninstall`	Open on removing Meelmo. With `--unattended` , remove it silently.
`--destdir ROOT`	Stage into a directory instead of the machine.
`--platform windows|linux`	The platform to lay down, with `--destdir` .

The Windows installer

`Meelmo-<version>-setup.exe` asks the same questions on its own pages:

- **What will this machine do?**
- **Where should this machine keep its data?**
- **Which server does this machine use?**
- **Who reaches this server?**
- **Who administers this server?**
- **Licence**

It then runs the same setup engine, so a role means the same thing on Windows and Linux. Its components page lists:

Component	Notes
Meelmo desktop application	Fixed: always installed.
Recorder service	Installs `recorder\` and a default `recorder-config.json` in the data folder, written only if none is there.

Component	Notes
Data-access server	Files only.
Web client	The browser HMI. With a Chromium browser on the machine, it also adds a Start menu shortcut that opens `http://127.0.0.1:8443` in its own window.
Server tools	`Meelmo.Migrate` and `Meelmo.WhoHolds`, in `tools\`.
Demonstration	
Database history providers	Only in a build where they are a separate component.

The task **Create a desktop shortcut** puts the Desktop (and the browser HMI shortcut) on the desktop.

The installer's language is English, German, Spanish, French, Italian, Japanese, Korean or Russian. It opens in the machine's language when it is one of those, and in English otherwise. The product itself then speaks all ten of its languages.

Folders.

Path	What
`C:\Program Files\Meelmo`	The Desktop, with `recorder`, `web`, `tools`, `demo` and `setup` beneath it
`C:\Meelmo\Data`	The data folder (asked for). The browser host keeps its own settings, screens and browser profiles in its `web` folder, and reads the licence and the site's authority from the data folder itself.

The data folder is deliberately visible, and deliberately not under `Program Files`. It is created writable by the machine's users, because the services run as `LocalSystem` and a person's Desktop does not.

Services. `MeelmoRecorder` (the recorder) and `MeelmoWeb` (the browser host), as the role requires. The data-access server is never registered by the installer.

Upgrading. Run the newer installer over the installed one. Windows recognises the package by an identifier that never changes, so it upgrades in place. The application, the services' programs and the demonstration plants are replaced; the data folder is not.

The services are stopped before their programs are replaced and started again afterwards, even when the installation is cancelled or fails. Close the Desktop if it is open. A silent upgrade (`/VERYSILENT`) without an answer file leaves the data folder and the registered services as they are.

Downgrading replaces the files the same way, and is not recommended. A plant's data written by a newer build is not guaranteed readable by an older one. Restore from backup instead.

If it cannot finish, the last page names what it could not do and where the log is (`<data folder>\logs`). Run the installer again: it opens on what the machine already is, and finishes.

Linux: where things go

Path	What
`/opt/meelmo`	The Desktop
`/opt/meelmo-recorder`	The recorder service
`/opt/meelmo-web`	The browser host

Path	What
`/opt/meelmo-tools`	The server tools
`/opt/meelmo-setup`	Meelmo Setup, kept for removal
`/usr/share/meelmo/demo`	The demonstration plants
`/var/lib/meelmo`	The data folder: configuration, node, recordings, the site's authority, `logs`
`/var/lib/meelmo-web`	The browser host's own settings, screens and browser profiles
`/etc/meelmo/installation.conf`	What this machine was set up as. It never holds a password.

Programs live under `/opt` and state under `/var/lib`. An upgrade replaces the first and leaves the second exactly as it is. The services run as the `meelmo` account.

systemd unit	Runs	State
`meelmo-recorder.service`	`/opt/meelmo-recorder/Meelmo.Recorder /var/lib/meelmo/recorder-config.json`	`MEELMO_STATE=/var/lib/meelmo`
`meelmo-web.service`	`/opt/meelmo-web/Meelmo.Web.Host`	`MEELMO_STATE=/var/lib/meelmo-web`
`meelmo-dataaccess.service`	The data-access server, on `127.0.0.1:7445`	`MEELMO_STATE=/var/lib/meelmo`. Shipped with the data-access server's own archive. No role registers it.

Where a machine runs more than one program, setup connects them: the browser host is made a client of the recorder's node through a drop-in under `/etc/systemd/system/meelmo-web.service.d/`. Use `systemctl status meelmo-recorder` and `journalctl -u meelmo-recorder` to watch a service. Each program also writes `logs/<program>.log` in its state folder (8 MB, with one older file kept as `.1`).

The recorder, the browser host and the data-access server also ship as archives of their own: the program, its systemd unit and a README, with no installer. They are for a site that packages Meelmo itself.

Containers

Meelmo builds three container images from `deploy/docker/Dockerfile`. They are for evaluation, edge deployment on Docker or k3s, and sites that accept only containerised workloads. They are not a substitute for the signed installer: an image carries no code signature. The Desktop is not containerised.

Image target	Contains	Port
`web`	The node, the browser HMI and Studio	8080
`dataaccess`	REST, gRPC, the Grafana data source and MCP	7443
`recorder`	Unattended recording	5077 (status API)

```
docker compose -f deploy/docker/compose.yaml up --build
# then open http://localhost:8080 and complete the first-run page
```

`compose.yaml` is an evaluation stack. The `web` container owns the node and is published on `127.0.0.1` only, with `MEELMO_WEB_ALLOW_INSECURE` set. That is an evaluation posture, and the host says so.

To record, add `compose.recording.yaml` on top. It inverts the stack: the recorder owns the node and the other two attach to it at `https://recorder:8444`. It needs a `recorder-config.json` and a certificate issued for the name `recorder`.

```
docker compose -f deploy/docker/compose.yaml -f deploy/docker/compose.recording.yaml up -d
```

- **One volume matters.** `/var/lib/meelmo` holds the node, its history, its secrets and its logs. Back it up.
- **Non-root.** Every image runs as uid 10001.
- **Health.** Each program accepts `--health-check`. It probes its own `/health` on loopback and exits 0 or 1.
- **TLS.** Off loopback, TLS is required. Supply `MEELMO_WEB_CERT`, or terminate TLS at a reverse proxy (`deploy/reverse-proxy` has configurations for nginx and stunnel).
- **The recorder is configured by its file**, not by the environment. Other containers are configured by the variables in The browser host.
- **Monitoring.** `deploy/monitoring` has a Prometheus scrape configuration, alert rules and a Grafana dashboard for the `/metrics` each container serves.

The Community Edge VM

The Community Edge VM is a complete Meelmo workstation on Ubuntu 24.04. It holds:

- the node, the browser HMI and Studio, and the Desktop;
- a simulated line already feeding it.

It is one person's workstation: one account, one live viewer at a time, 32 tags, and no recorder. The *Community Edge VM Setup Guide* covers hypervisors, the host clock and networking in detail.

What it needs. UEFI firmware with Secure Boot off, 2 vCPUs, 4 GB RAM and a bridged network. Import the `.ova` into VMware Workstation, Player, ESXi or vSphere, or the `.qcow2` into QEMU/KVM or Proxmox. Check the file's SHA-256 against your download page first.

First start, at the VM's console. It asks two things, once:

1. **Your account.** A username and a password, typed twice. This one account is both the Linux login (with `sudo`) and the **Meelmo administrator** for the Desktop and the browser HMI.
   - The password needs at least 12 characters and three of lowercase, uppercase, digit and symbol.
   - `meelmo` is reserved for the product's own service.
   - There is no default password.
2. **Your time zone**, as a name such as `Europe/Rome` (any capitalisation). **Enter** keeps UTC. Clocks, logs and the HMI show this zone; data is always recorded in UTC.

The VM then installs itself, with no browser wizard and no key. The console prints one line per step:

Line	Step
`phase=clock`	Waits up to 90 seconds for a time sync
`phase=setup`	Installs Meelmo (a few minutes)
`MEELMO-VERIFY: … =ok`	Checks the installation
`phase=pki`	Makes the site's certificate authority
`phase=network-config`, `hmi-start`, `root-fingerprint`	Opens the HMI on the network and prints the root fingerprint and the HMI address

Line	Step
`phase=idle-sign-out`	Turns idle sign-out off
`phase=hmi-example`	Installs the *Example line* screen
`phase=desktop-sign-in`	Signs the Desktop in with your account

Write down the root fingerprint and the address (`https://meelmo-edge.local:8443/` , or `https://<VM-address>:8443/` behind NAT). The VM then shows the graphical login.

Using it. Sign in with your account. The desktop has two launchers:

- **Meelmo HMI** opens the browser HMI at `https://127.0.0.1:8443/` , on **Example line**.
- **Meelmo Community** opens the Desktop, already signed in to the node.

From another PC. Open `https://meelmo-edge.local:8443/trust` and compare the fingerprint with the console's. **Only if they match**, install the root certificate as a trusted root. Then open `https://meelmo-edge.local:8443/` .

Other Linux users on the VM. The first time another Linux user opens **Meelmo Community**, it asks for a Meelmo sign-in. The Desktop then joins the node as that user's workstation; the password is not kept. **Cancel** skips the question. You can sign in later from **Administration ▸ Deployment…** by entering the user and password and pressing **Save**. Until then the Desktop reports *Historian: unavailable*.

Limits.

- Live values stream to one viewer at a time. A second browser shows its values as *Disconnected*, and takes the stream over within about a minute of the first one closing.
- History is off until you enable the embedded historian.
- Idle sign-out is off on this VM. Closing the browser or the Desktop still signs you out.

Repairing. Sign in on the text console (**Ctrl+Alt+F2**) and run `sudo meelmo-commission` . It reinstalls Meelmo over the existing data and keeps the site's certificate authority. If the console stops at `MEELMO-HANDOFF: failed phase=…` , reboot the VM: it asks for your password again and retries.

Running the programs by hand

The installer registers the programs as services, and you rarely need more. This section is for a site that runs them itself: in a container, from a script, or while diagnosing.

Where each program keeps its state. Every Meelmo program finds its state folder the same way. The environment variable `MEELMO_STATE` wins if it is set; otherwise it is `state` beside the executable. Each program writes the folder it chose, and why, as the first line of its log. Two programs that disagree about this path are two installations on one machine.

The recorder service (Meelmo.Recorder)

```
Meelmo.Recorder [path-to-recorder-config.json]
Meelmo.Recorder --health-check [path-to-recorder-config.json]
```

With no path, it reads `recorder-config.json` beside the executable. It runs:

- in a console (stop it with `Ctrl+C`);
- as the Windows service `MeelmoRecorder` ;
- as a systemd unit of `Type=notify` .

A fatal start ends the process with exit code `2`. Examples are an unlicensed installation, a node that cannot be opened, and a lease connection string that is not one. A service manager therefore sees a failure and restarts it. `--health-check` probes the status API on the port the file names (5077 if it cannot be read) and exits `0` or `1`. It answers only when the status API is switched on.

Recording ▸ Recorder Service… on the Desktop writes this file for you (see the *Recording and playback* chapter). The file accepts comments and trailing commas. Its fields:

Field	Default	Meaning
`nodeDirectory`	Empty: the `node` folder beside this file	The node this service holds. What it acquires, its alarms and its recording preferences come from that node's activated revision; there is no project file here.
`controlPlaneHost`	`127.0.0.1`	Where clients reach the node's control plane
`controlPlanePort`	`8444` in the shipped file; `0` turns it off	Its port
`controlPlaneCertificatePath`, `controlPlaneCertificatePasswordSecret`	—	A certificate for it. A routable bind without one is refused.
`controlPlaneRequiresClientCertificate`	`false`	Also require an enrolled machine certificate. Enrol the machines first.
`outputDirectory`, `filePrefix`	`recordings`, `recording`	Where rotated `.opslog` files go, and their name
`maxFileMegabytes`, `rotationMinutes`	`256`, `60`	Rotate at this size or age (`0` turns either off)
`retentionDays`	`7`	Delete rotated files older than this (`0` keeps them)
`compressRotatedFiles`	`true`	Gzip rotated files to `.opslog.gz`
`lossless`	`false`	Never drop a sample; acquisition waits for the writer
`flushSeconds`, `writeJournal`, `channelCapacity`	`5`, `true`, `50000`	Flush interval, the recording journal, and the queue between acquisition and the writer
`includeTopics`, `excludeTopics`	Empty	Glob lists over `connection/group/tag`. An empty include records everything.
`triggerEnabled`, `triggerCondition`, `triggerHoldSeconds`, `triggerPreBufferSeconds`	Off, empty, `5`, `0`	Record only while a condition over `[connection/group/tag]` is true, with a hold after and a buffer before
`dataAccess`	Off	`enabled`, `host` (`127.0.0.1`), `port` (`7443`), `grpcEnabled`, `grpcPort` (`0` shares the port), `mcpEnabled`, `certificatePath`, `certificatePasswordSecret`, `allowInsecureRemote`, `trustedProxies`, `stateDirectory`
`acquisitionServer`	Off	`enabled`, `listenAddress` (`127.0.0.1`), `listenPort` (`9701`), `localTransport` (`true`), `certificatePath`, `certificatePasswordSecret`, `requireClientCertificate`

Field	Default	Meaning
`sitePki`	`manage: true`	`manage` creates and renews the site's own certificate authority on first start. Set it to `false` where certificates come from the site's own PKI. `siteName` names the authority.
`apiEnabled`, `apiHost`, `apiPort`, `apiToken`	Off, `localhost`, `5077`, empty	The read-only status API. A routable bind needs a token.
`allowInsecureApi`	`false`	Permits a routable status API without a token. Use a token instead.
`standby`, `promotedBy`, `promotionMarkerFile`, `lease`	Off	A redundant pair. See A redundant pair.

Rotation, retention, compression and losslessness are the node's, edited on the recording screen. The values here are used only before a node has activated anything. Where history is stored is also the node's (**Data ▸ Historian…**). A `historian` block left in an older file is reported at start-up and not read.

The browser host (Meelmo.Web.Host)

```
Meelmo.Web.Host [--node [DIRECTORY | https://host:port]] [--allow-insecure]
```

Option	Meaning
`--node`	Serve a node. On its own, it means the `node` folder in this host's state folder. A directory is a node this host holds. An `http(s)://` address is a node elsewhere that this host is a client of. Without it, the host does what its first-run page saved.
`--allow-insecure`	Permits serving a routable address without TLS. Correct only behind a TLS-terminating proxy on a trusted network. The host says so on every start.
`--seal-secret`	Reads one line from standard input, prints it sealed with this state folder's key, and exits. This is how you produce a value for a field that refuses cleartext, such as the single sign-on client secret.
`--initialize-appliance-pki`	Used by the Community VM's commissioning: adopts a Meelmo-managed site authority if none is managed yet, and exits.
`--health-check`	Probes `/health` on `MEELMO_WEB_PORT` (8080 if unset) and exits 0 or 1.

Its settings live in `web-config.json` in the state folder, written by the first-run page and the administration screens. The environment overrides the file where both name a setting:

Variable	`web-config.json`	Meaning
`MEELMO_STATE`	—	The state folder
`MEELMO_NODE`	`acquisition.useNode`, `acquisition.nodeDirectory`	As `--node`
`MEELMO_NODE_ALSO`	`acquisition.alternateNodes`	The other machines of a redundant pair, as `http(s)` addresses separated by `,` or `;`
`MEELMO_NODE_TOKEN`	`acquisition.nodeDataTokenSecret` (sealed)	This host's credential for the node's data API. Without one it serves no history, and says so once at start.

Variable	web-config.json	Meaning
`MEELMO_WEB_HOST`, `MEELMO_WEB_PORT`	`network.host`, `network.port`	Bind address and port (`127.0.0.1`, `8443`)
`MEELMO_WEB_CERT`, `MEELMO_WEB_CERT_SECRET`	`network.certificatePath`, `network.certificatePasswordSecret`	The TLS certificate (PFX). Its presence turns TLS on.
`MEELMO_WEB_ALLOW_INSECURE`	`network.allowInsecure`	As `--allow-insecure`
`MEELMO_WEB_TRUSTED_PROXIES`	`network.trustedProxies`	Reverse proxies whose `X-Forwarded-For` is believed, as addresses or CIDR networks. An entry that is not one stops the start.
`MEELMO_WEB_PUBLIC_ADDRESS`	`network.publicAddress`	The address browsers reach this host at (see *Configuration and deployment*)
`MEELMO_WEB_SERVE_NODE`	`network.serveNode`	Serve the node this host holds to other workstations, with live data on `network.nodeCarrierPort` (`9701`)
`MEELMO_WEB_ALLOW_WRITES`	—	`true` permits writes to the plant from the browser, for users holding the command permission. Off by default.
`MEELMO_HMI`	—	Where the screens are kept (`hmi` in the state folder)
`MEELMO_HISTORIAN`	—	Moves the node's history off the state folder, for every program at once
`MEELMO_LICENSE_FILE`, `MEELMO_LICENSE`	—	A licence supplied as a mounted file or as the value itself. It is never copied into the state folder.

A configuration file that cannot be read stops the host with exit code `2`.

The Desktop

```
Meelmo --state DIRECTORY
```

`--state` points the Desktop at another state folder, exactly as `MEELMO_STATE` would. Use it to open a second, separate installation's state, for example a copy restored from backup, without touching the machine's own. The Desktop's other command-line modes are used by setup and are not for you to run.

The data-access server

`Meelmo.DataAccess.Server` is described with its options in the *Data access and SDKs* chapter.

A redundant pair

A **standby** is a second recorder service that waits while another one holds the node, and takes the plant over when the holder stops. It is set in `recorder-config.json` only. No screen offers it.

What a standby is not. There is no warm node redundancy.

- **A takeover is a restart, not a handover.** Acquisition starts cold, the acquisition counters begin again at zero, and standing alarms are annunciated again with their acknowledgements forgotten; shelved alarms survive. Nothing records the plant while no host holds it. Every start is written into the node's trail, saying whether it waited and whether the previous holder released.
- Meelmo ships **no heartbeat, quorum or peer discovery**. A standby never decides for itself that the other host has died.
- Across machines, the database lease's write path **does not yet carry its fencing token**.

If you need automatic failover of acquisition between machines without a cluster manager, this product does not have it.

On one machine

This covers the recorder process: a crash, running out of memory, or a person killing it.

1. Give the second service its own configuration file, **in the same folder** as the first's, with the same `nodeDirectory`.
2. Set `"standby": true` in it. Leave `promotedBy` at its default, `"thisMachine"`.

The operating system drops the holder's hold the moment the process ends. The standby's next attempt succeeds, with no timeout and no lock file to delete. Keep the two files in one folder (or set `MEELMO_HISTORIAN` for both): history is placed relative to the configuration file, and a standby configured elsewhere would record into a historian of its own.

Leave `standby` false on the service that owns the node. A second recorder *without* `standby` is refused and exits, which under a restarting service manager is a crash loop against a healthy primary.

Across two machines

Two machines cannot share one node folder. An exclusive hold on a network filesystem does not exclude the other machine, so Meelmo refuses a node folder that cannot be held. There are three arrangements:

Arrangement	How	What it trusts
A cluster manager	A shared failover volume owned by a cluster manager that fences (Windows Failover Clustering, Pacemaker), with the standby waiting on the mount	The cluster manager
`"promotedBy": "theLease"`	Both machines have their own copy of the same node, under the same node folder **path**. Both set `lease.enabled: true` and the same PostgreSQL `lease.connectionStringSecret`; the second also sets `standby: true`.	One statement in a PostgreSQL database both machines reach. It need not be the historian's database. SQL Server and Oracle connection strings are refused at start-up.
`"promotedBy": "anOperator"`	The standby waits until a person creates the file named by `promotionMarkerFile` (default `promote`, beside the configuration file)	A person who has established that the other host has stopped

What the lease costs.

- While the database cannot be reached, a running holder keeps acquiring until its term (`lease.termSeconds`, default `30`) runs out. It then **stops**, and no other machine may start.

- A takeover cannot be quicker than the term, because a dead machine released nothing.

Shortening the term speeds a takeover and shortens, by exactly as much, the database outage a running plant tolerates. Thirty seconds is a starting point, not a recommendation.

Promotion by a person.

- Whatever you write in the marker file is recorded in the node's audit trail, so write why.
- The file is renamed when it is acted on.
- **A promotion is not remembered.** Afterwards, set `standby` to `false` on this machine, or the next restart makes it a waiting standby again.

`promotedBy` set without `standby: true`, or `theLease` without `lease.enabled`, is refused at start-up rather than ignored.

Watching a pair

Switch the status API on (`apiEnabled`) on both hosts. A standby answers while it waits. `/status` carries:

- `standby`: whether this host is the waiting half of a pair;
- `holding`: whether this host is the one acquiring the plant.

In `/metrics` these are `meelmo_recorder_standby` and `meelmo_recorder_holding`.

Alarm on `holding` summed across the hosts of one plant: it must be exactly 1.

- Zero means nobody is acquiring. With promotion by a person, that alarm is the only thing that will tell anyone to act.
- Two means two machines are polling one plant.

Do not alarm on `meelmo_recorder_up`. It is 0 on a standby doing exactly what it was asked.

Pointing clients at whichever machine holds the plant

Redundancy decides which host acquires. It says nothing about where people are looking. Name the other machine in each client:

- **Desktop:** `deployment.alternateHosts` in the Desktop's `settings.json`, as host names. The ports are the same as the main server's.
- **Browser host:** `acquisition.alternateNodes` in `web-config.json`, or `MEELMO_NODE_ALSO`, as `https://host:8444` addresses.

When a client attaches, it asks every named machine and attaches to the one that says it still holds the node. It never takes whoever answers first, because a standby answers too, and so does a host that has been deposed.

Which machine holds it: Meelmo.WhoHolds

`Meelmo.WhoHolds` asks the machines of a pair the same question a workstation asks, and prints the answer. It also checks the commissioning question underneath: whether the machines given are one plant at all. A pair is one commissioned node copied onto a second machine, so both carry the same node id. Two machines commissioned separately are two plants. The installer's **Server tools** part installs it beside `Meelmo.Migrate`: in `tools` under the installation folder on Windows, and in `/opt/meelmo-tools` on Linux. Run it from either machine of the pair.

```
Meelmo.WhoHolds <host> [host…] [--port 8444] [--tls] [--node <node-id>] [--pinned <root.pem>]
```

Option	Meaning
`<host> …`	The machines this plant may be running on. The first is the one the others are measured against.
`--port N`	The control-plane port (`8444`)
`--tls`	Ask over HTTPS
`--node ID`	The node id the machines must carry
`--pinned FILE`	The site's root certificate, as PEM, to trust

Exit code	Meaning
`0`	Exactly one machine holds the plant. Its name is the last line.
`1`	Nobody holds it: the plant is not being acquired.
`2`	More than one machine claims it.
`3`	The arguments were wrong.
`4`	The machines given are not one plant, so they are not a pair.

Worked example: a standby on the packaging line's server

`pkg-srv-01` holds `PKG-SRV-01` with the recorder service. The plant wants acquisition back within seconds if the recorder process crashes.

1. Copy `/var/lib/meelmo/recorder-config.json` to `/var/lib/meelmo/recorder-standby.json`, in the same folder.
2. In the copy, set `"standby": true`. Turn the status API on in both files (`"apiEnabled": true`), on port `5077` in the first and `5078` in the copy.
3. Register a second service that runs `Meelmo.Recorder /var/lib/meelmo/recorder-standby.json`, like the first, and start it. Its log says it is a standby for the node, promoted by this machine, and `/status` on port 5078 reports `standby: true, holding: false`.
4. Set an alarm in the plant's monitoring for `sum(meelmo_recorder_holding)` across both ports ≠ 1.
5. To test it, kill the main recorder (`kill -9`). Within moments `holding` moves to the standby. The node's trail records the start, and that the previous holder did not release. Standing alarms reappear unacknowledged; acquisition counters restart at zero. The sum of `holding` was 0 only for the gap.
6. Do not simply restart the main service: without `standby` it is refused while the standby holds the node, and exits. Set `"standby": true` in its file first. It then starts as the waiting half, and the two have swapped roles.

This protects against the recorder process failing. It does not protect against `pkg-srv-01` itself failing; for that, see Across two machines.

Control loops

A plant has hundreds of PID loops, and nobody looks at them until one misbehaves badly enough for an operator to put it in manual. By then it has usually been cycling for weeks. The loop monitor reads every declared loop from the historian on a schedule and answers, for each, three questions: **is it doing its job, what is stopping it, and what should be done about it** — and it lists the loops worst first, so the morning goes to the ones that matter.

This chapter covers:

- declaring a loop, and having the tag names propose loops for you;
- what the monitor checks, and the methods behind each check;
- reading a verdict, the shared-cycle panel, and a loop's evidence;
- the schedule, the indices it keeps, and alarming on them;
- the routes and the assistant tool;
- what it cannot tell you.

Everything here reads the historian, so the loop's signals must be recorded — and recorded well enough. The first thing the monitor checks is whether they are (see *The data comes first*).

Declaring a loop

A loop is declared in the project, in the semantic section's loop document. Nothing is monitored that is not declared: the monitor never guesses which signals form a loop.

Field	Meaning
`id`	The loop's tag, e.g. `FIC-101`. Required.
`type`	`flow`, `pressure`, `temperature`, `level`, `averagingLevel` or `other`. A level loop is judged as the integrator it is; an averaging level is meant to swing, so its tracking and variability are not held against it and it may cycle five times wider before that counts.
`pv`	The measurement's signal. Required.
`sp`	The setpoint's signal. Without it the loop is judged on its variability about its mean.
`op`	The controller output's signal, in percent. Required.
`mode`	The mode signal, when the controller publishes one. `manualModes` lists the values that mean manual (default `MAN`, `MANUAL`, `M`, `0`). Without a mode signal, manual is inferred and said to be.
`valvePosition`	The valve's measured position, when there is one. With it, stiction is read directly from the gap between output and stem.
`valve`	The final element's name, used in the sentences ("FV-101 is likely sticking").
`pvLow`, `pvHigh`	The measurement's range. Every percentage is of this range; without it the range is read from the data and the verdict says so.
`opLow`, `opHigh`	The output's range (default 0–100).
`timeConstantSeconds`, `deadTimeSeconds`	The process's response, when known. The dead time makes the minimum-variance index meaningful; the time constant lets the data check say whether the sampling is fast enough.
`enabled`	`false` keeps the loop declared and stops assessing it.

The document also carries a `schedule`: `enabled`, `everyMinutes` (default 60, at least 15) and `windowHours` (default 8, at most 168).

Editing the loops on the Desktop

Data ▸ Control loops… ▸ Edit the project's loops… opens the loop document itself, or **Edit…** beside a declared loop's verdict opens it on that loop. On the left is the list of the loops the project declares (tag, type, name, and *not assessed* for a loop switched off), **New loop** and **Remove loop…** under it, and the **Schedule**: **Assess the loops on a schedule**, **Every (minutes)** and **Over the last (hours)**. On the right are the chosen loop's fields, in the order of the table above:

- **Tag**, **Name**, **Type** and **Assess this loop**;
- **Signals** — **Measurement (PV)**, **Controller output (OP)**, **Setpoint (SP)**, **Valve position**, **Mode** — each a box that offers the project's signals as you type and takes one it does not offer; **Mode values meaning manual**, separated by commas (empty uses `MAN, MANUAL, M, 0`); and **Valve tag**;
- **Ranges and dynamics** — **PV low**/**PV high** (both, or neither to take the range from the data), **OP low (%)**/**OP high (%)** (empty is 0 and 100), **Time constant (s)** and **Dead time (s)**.

Keep loop puts the fields into the list — replacing the loop they were loaded from, in its place even when its tag was changed, or adding a new loop at the end. It refuses a loop the document's own check refuses, in the same words the node uses (*The loop 'FIC-101' gives one end of its measurement range; give both, or neither…*), a tag another loop already has, and a number written with a comma. **Remove loop…** asks the shared delete question; *Disable instead* keeps the loop switched off. **Save** checks the schedule and the whole document and puts it into the project; **Cancel** leaves the project as it was. Either way the node assesses the change once the project is applied. A loop document this build cannot read is reported and never saved over.

Proposed from the tag names, never assumed

Loops ▸ Loops the tag names suggest (web) and **Propose loops from tag names** (Desktop) read the catalogue for tags written as one name with a parameter suffix — `.PV` / `.CV` / `.MEAS`, `.SP` / `.SV`, `.OP` / `.OUT` / `.MV` / `.CO`, `.MODE`, `.POS` / `.ZT` — and propose each group that has at least a measurement and an output as a loop, with the type read from the ISA first letter (`F`, `L`, `P`, `T`) where the name has one. Each proposal says which suffix every signal was read from. A suffix is a convention, not a fact, so a proposal goes into the project only when an engineer keeps it — on the web through the same reviewed project draft a model or a rule goes through; on the Desktop with **Add to the project**, after which the project is applied as usual. **Remove from the project…** on the Desktop asks first and offers to switch the loop off instead.

The data comes first

A loop whose data cannot carry a conclusion is reported as **Data not usable**, with the reason, rather than judged. A wrong "your valve is sticking" costs a maintenance crew a morning; "the data cannot say" costs nothing. The check, before any method runs:

Problem	Effect
No readings, fewer than 200 usable samples, too little of the window covered, or no automatic period long enough	Not usable. A stretch the historian reports it has nothing for — the node stopped, a source was unreachable — is a gap: no value is held across it, however short, so it counts against the coverage rather than as steady data.
Readings marked bad, or sentinel values such as −9999 (the quality checks the historian already runs)	Not usable past a share; otherwise noted.

Problem	Effect
Quantization — fewer than 8 distinct values	Not usable. Under 30 levels, or a step of 1 % of range or more, the shape tests are not trusted and confidence is halved.
Compression — straight lines between stored points (interpolated), or changes only in jumps (exception recording), or one reading kept in ten	Not usable at ten or more; confidence reduced from 1.5.
A frozen transmitter — the measurement does not move while the output does	Not usable.
Sampling slower than the process — the sampling period longer than the time constant	Not usable; slower than a quarter of it reduces confidence. Without a declared time constant, the typical response of the loop's type is used and the verdict says so.
Stale or spiky readings	Noted, and the spikes left out.

The action for every one of these is the same: store the loop's signals uncompressed, at full resolution, at least five times faster than the loop responds.

What is checked, and how

Each criterion produces a finding with a rating (good, fair, poor), a badness from 0 to 1 that can be compared across loops, a confidence, one sentence with its numbers, and an action. The loop's status is its worst finding; the limiting criterion is the one that set it.

Oscillation

The error's autocorrelation is computed and its zero crossings examined (Thornhill, Huang and Zhang, 2003). A regular oscillation has evenly spaced crossings: the regularity is the mean period over three times its spread, and a value above 1 over at least three cycles is a regular oscillation. The amplitude is the median half-swing of the band-passed error. A set point that itself cycles at the same period is not counted against the loop.

Alongside it runs Hägglund's (1995) load-disturbance detector: the error's integral between zero crossings against a limit of 2a/ω_u, with ten disturbances in fifty ultimate periods meaning the loop is oscillating. It catches irregular cycling the autocorrelation does not, and is reported only where the autocorrelation agrees the cycling is irregular — on its own it flags healthy loops with random load too often.

Stiction

A sticking valve makes a loop cycle; so do aggressive tuning and an outside disturbance. The cure differs — a valve overhaul, a retune, or nothing in this loop at all — so stiction is never called on one test. Only a loop that oscillates is assessed, and at least two independent tests must agree:

- **Cross-correlation** (Horch, 1999): for a self-regulating loop, the PV–OP cross-correlation is odd under stiction and even under tuning or disturbance.
- **Second difference of the level** (Horch's variant for integrating loops): under stiction the level's second difference is spiky; its kurtosis is the test.
- **Shape** (He et al., 2007): a sine and a triangle, each with a free apex, are fitted to each half-cycle of the measurement (self-regulating) or the output; triangles mean stiction.
- **Nonlinearity** (Choudhury, Shah and Thornhill, 2004): the bicoherence's non-Gaussianity and nonlinearity indices. Supporting only — it says the cycle is nonlinear, not why.

- **Valve position**, when declared: half the spread of the gap between output and stem is the stick band, read directly.

The combined confidence is **Likely** at 0.6 and **Possible** at 0.4. When the tests disagree it is capped below Likely, and a sinusoidal, linear cycle is capped below Possible. Quantized data reduces the shape test's weight. A Likely stiction finding names the valve and estimates the apparent stiction (twice the output's amplitude times the sine of the PV–OP phase). A Possible one asks for the decisive test instead: put the loop in manual for a few cycles — if the cycling stops, test the valve; if it carries on, look upstream.

Saturation and windup

The share of time the output spent at either limit, and the longest stretch. An output that stays at its limit after the error has changed sign is integral windup, and the action says to check the anti-windup.

Manual

The share of time in manual, from the mode signal when there is one; otherwise inferred from an output that stays put while the measurement moves, and said to be inferred.

Valve travel

Reversals and travel a day, in percent of stroke, counted through a half-percent backlash so noise does not count as travel. High travel is attributed to its cause: the oscillation or stiction already found, or the measurement noise the controller is passing to the valve.

Set-point tracking

Mean absolute and root-mean-square error in percent of range, and for each set-point step the settling time (the first entry into a band that the measurement then stays in) and the overshoot.

Variability and the minimum-variance index

The error's standard deviation in percent of range, and the Harris index (Harris, 1989; Desborough and Harris, 1992): the share of that variance that a minimum-variance controller could remove, from a time-series model of the error and the loop's dead time. The dead time is the declared one, or estimated from output steps; the index is reported as **not reliable**, with the reason, when the dead time is unknown or within one sample, the window is short, or the set point moved.

Sensor noise

The measurement's high-frequency noise, estimated robustly from its second difference, in percent of range.

Shared cycles: one fault, many loops

When several loops cycle at the same period (within 12 %), they are grouped, and the group names its likely source: the loop with the strongest stiction evidence, else the one whose cycle is least sinusoidal (nonlinearity is strongest at the source and filtered downstream), else the one leading in phase. The followers' verdicts say they are following it, and their action is to fix the source first. On the web the groups are a panel above the list.

Reading the screens

Web: Analytics ▸ Loops. The list opens on the node's latest scheduled assessment, with when it ran, over what window, and when it runs next. Each row is the loop, its status in a word and a colour, what limits it, the verdict with its action, and the confidence. **Assess now** asks for the window in the form instead; **Show the node's latest assessment** goes back.

A loop's own page (click its tag) re-assesses it with its evidence: the verdict and any data problem; the measurement and set point, and the output, over the window; the error's autocorrelation with its zero crossings; the measurement against the output (an ellipse for a linear loop, a parallelogram under stiction); the output's histogram (piled against a limit under saturation); every criterion in a table; and the indices over the weeks. Each chart has a sentence under it saying what it shows. The trends break where the history has a gap, as the main trends do: a node restarted for two minutes is two minutes of nothing, not a flat line.

Desktop: Data ▸ Control loops… shows the same list, and one loop's verdict and evidence beside it when it is selected.

The schedule, and the indices it keeps

The node assesses its enabled loops every `everyMinutes` over the last `windowHours` (hourly over eight hours by default) and keeps the result, so the list is ready when the page opens. Each assessment publishes its indices as health signals, recorded by the historian like any other signal:

`Health/Loops/<loop>.<index>`, for the indices `Status` (0 good, 1 fair, 2 poor, 3 data not usable), `Badness`, `Confidence`, `StictionConfidence`, `OscillationAmplitude`, `OscillationPeriod`, `LoadDisturbancesPerHour`, `Saturation`, `Manual`, `ReversalsPerDay`, `TravelPerDay`, `MeanAbsoluteError`, `Variability`, `Harris` and `Noise`.

Because they are signals, everything that works on a signal works on them: trend `StictionConfidence` over the months to see a valve wear; set an alarm rule on `Health/Loops/FIC-101.Status` reaching 2 to be told when a loop turns poor; base an advisory on them; include them in a report.

Over the API and the assistant

Route	What it does
`GET /api/v1/loops`	The node's latest scheduled assessment.
`POST /api/v1/analysis/loops`	An assessment now, over `start` – `end`, of all loops or those named in `loops`; `definitions` assesses loops not declared on the node; `evidence: true` adds the series a loop page draws.
`GET /api/v1/loops/{id}/kpis`	One loop's indices over time (six weeks unless `from` / `to` are given).
`GET /api/v1/loops/proposals`	Loops the tag names suggest.
`POST /api/v1/project/loops`	Stages a loop into the project draft for review.

All need the `ReadHistory` permission, and a loop one of whose signals the caller may not read is left out of every answer and counted, never described. The assistant's tools `analysis_loops` (assess now) and `analysis_loops_scheduled` (the latest assessment) answer the same questions.

What it cannot tell you

- **It reads closed-loop data only.** It cannot separate a sticking valve from a nonlinear process with certainty — that takes a valve test — which is why a Possible verdict asks for one.
- **A loop that never moves is hard to judge.** A loop at steady state with a quiet process shows little; good means nothing stands out, not that the tuning is optimal.
- **Percentages need a range.** Declare `pvLow` and `pvHigh`; numbers against a range read from the data are not comparable between loops.
- **The minimum-variance index needs the dead time.** Without a declared or estimable one it is reported as unreliable rather than guessed.

A demonstration

`examples/loop-monitoring-demo.meelmoproj` simulates ten loops of a cooling section — one healthy, and one each with a sticking valve whose cycle two other loops follow, aggressive tuning, saturation, manual operation, a noisy transmitter, an outside disturbance, and a sticking valve with a position feedback. Open it, let it record for an hour or so, and open the Loops page: every finding above appears once.

References

- Choudhury, M. A. A. S., Shah, S. L., Thornhill, N. F. (2004). Diagnosis of poor control-loop performance using higher-order statistics. *Automatica* 40(10).
- Choudhury, M. A. A. S., Thornhill, N. F., Shah, S. L. (2005). Modelling valve stiction. *Control Engineering Practice* 13(5).
- Desborough, L., Harris, T. (1992). Performance assessment measures for univariate feedback control. *Canadian Journal of Chemical Engineering* 70(6).
- Hägglund, T. (1995). A control-loop performance monitor. *Control Engineering Practice* 3(11).
- Harris, T. J. (1989). Assessment of control loop performance. *Canadian Journal of Chemical Engineering* 67(5).
- He, Q. P., Wang, J., Pottmann, M., Qin, S. J. (2007). A curve fitting method for detecting valve stiction in oscillating control loops. *Industrial & Engineering Chemistry Research* 46(13).
- Horch, A. (1999). A simple method for detection of stiction in control valves. *Control Engineering Practice* 7(10).
- Thornhill, N. F., Huang, B., Zhang, H. (2003). Detection of multiple oscillations in control loops. *Journal of Process Control* 13(1).

The browser client: finding your way

> **Module: Web HMI.** This chapter describes the frame around every screen of the browser client: the top bar, the menus, the command palette, the status bar and the settings that belong to you. It is present only on installations licensed for the Web HMI module. The chapters after it (the operator HMI, the HMI Studio, automation, reports and analysis, and administration) all assume what is described here.

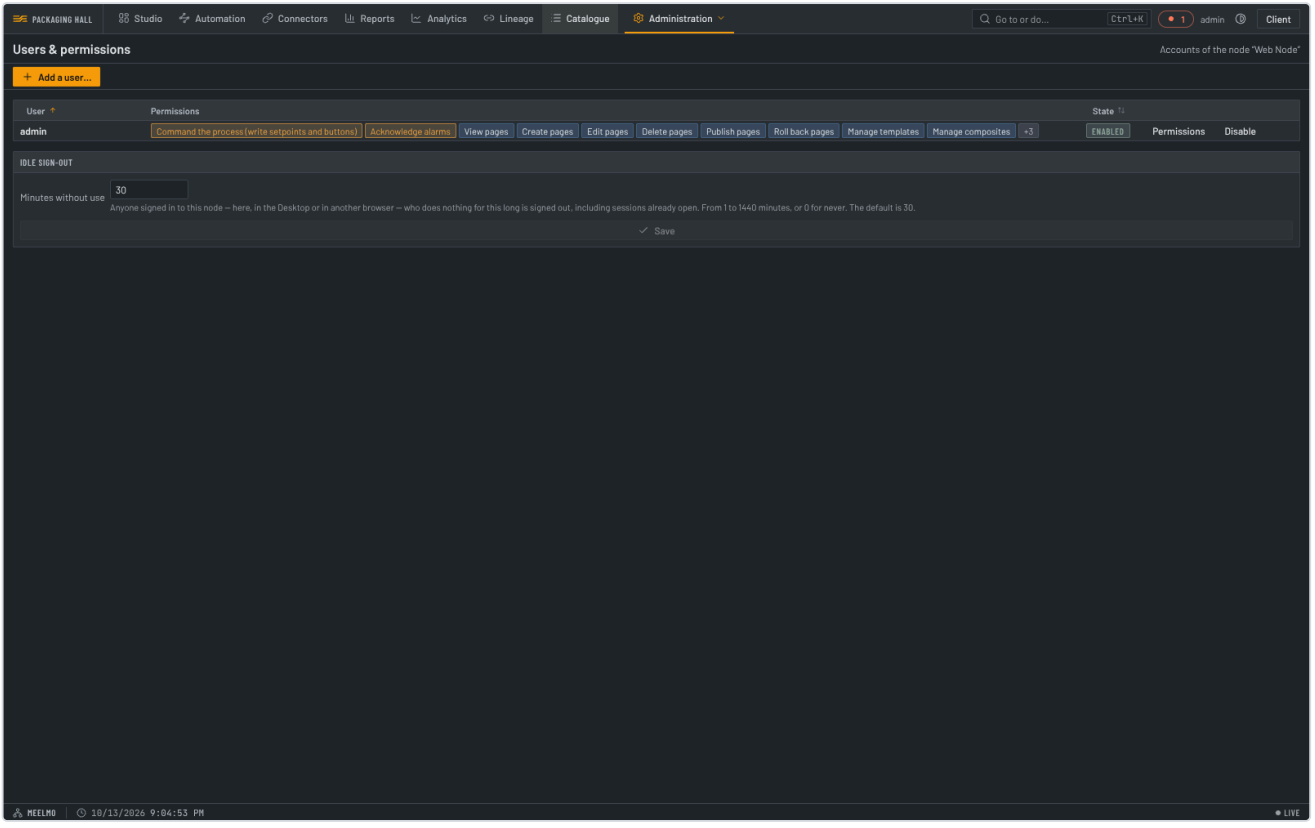
The browser client has two frames:

- **The operator screens** (`/hmi`). These show the plant: the page rail, the alarm banner and the operator's status bar. *The browser operator HMI* describes them.
- **The shell.** This frame holds everything else: the Studio, Automation, Connectors, Reports, Analytics, Lineage, the Catalogue and the administration screens. It has one top bar and one status bar, and they look the same on every screen. This chapter describes it.

You move between the two with **Client** (from the shell to the operator screens) and with **⋯ ▸ Go to** (from the operator screens to the shell).

The examples use the demonstration packaging line: node **Web Node**, the site named **Packaging hall** in the bar, and the seeded accounts `admin` (Administrator) and `shift` (Operator).

The top bar



The shell's top bar for the admin account on the packaging hall: the site name, the destinations Studio, Automation, Connectors, Reports, Analytics, Lineage and Catalogue, the Administration menu (lit, because Users & permissions is showing), the Go to or do... box with Ctrl+K in it, the alarm indicator showing one alarm, the user name, the theme button and Client. At the foot, the status bar with the site, the date and time, and LIVE.

From left to right:

Item	What it is for
Mark and site name	The site's own mark, or the product's when the site has supplied none, and the name this installation gave itself (here **Packaging hall**). Pressing it opens the client's root address, which shows your start page (see *Your own settings* below)
Work destinations	One entry per place you can work in, each an icon and a word. The place you are in is underlined, including any page inside it: an open Studio page still lights **Studio**
Administration	A menu of the administration screens, offered only to somebody who may open at least one of them. See *The Administration menu*
Go to or do…	Opens the command palette. **Ctrl+K** is written in the box
Alarm indicator	Whether anything is standing in alarm on this installation. See *The alarm indicator*
Your name	The display name of the person signed in
Theme button	Switches the theme. Absent where the project does not let people switch
Client	Opens the operator screens (`/hmi`), whatever your start page is

The browser tab is named after the destination you are in and the site, for example `Reports · Packaging hall`, unless the screen names itself.

The work destinations

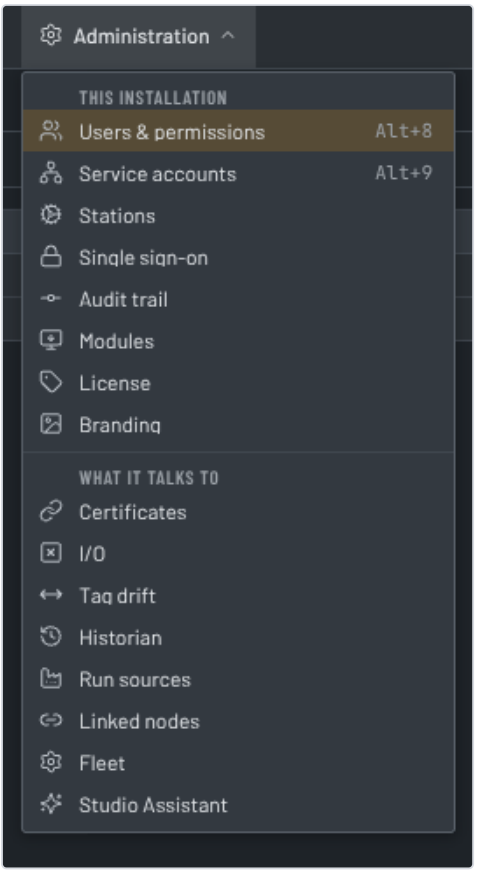
The bar offers only what your session may open. A destination whose screens would all refuse you is left out.

Destination	Address	Who sees it
Studio	`/studio`	People with the **Edit pages** permission
Automation	`/automation`	People the node grants `Automation.Read`. Not offered on a host that has no node to run rules on
Connectors	`/automation/connectors`	The same as **Automation**
Reports	`/reports`	People with `Reports.View`
Analytics	`/analytics`	Everybody. The hub then offers each tool only to those who may use it
Lineage	`/lineage`	Everybody. Each person sees only the part of the plant they may read
Catalogue	`/catalogue`	Everybody. The search covers only the signals you may read

Example. Signed in as `admin` on the packaging line, the bar shows all seven destinations. An account without **Edit pages**, `Automation.Read` or `Reports.View`, and with no administration screen open to it, sees only **Analytics**, **Lineage** and **Catalogue**, and no **Administration** menu.

The Administration menu

Administration opens a menu in two groups. The group headings are captions; each line is a link, so a middle click opens the screen in a new browser tab. The screen you are on is marked, and the **Alt+N** shortcut a line answers to is written beside it.



The Administration menu open over the Reports screen: This installation with Users & permissions (Alt+8), Service accounts (Alt+9), Stations, Single sign-on, Audit trail, Modules, License and Branding, then What it talks to with Certificates, I/O, Tag drift, Historian, Run sources, Linked nodes, Fleet and Studio Assistant.

Group	Screen	Address
This installation	**Users & permissions**	`/admin/users`
	Service accounts	`/admin/service-accounts`
	Stations	`/admin/stations`
	Single sign-on	`/admin/single-sign-on`
	Audit trail	`/admin/audit`
	Modules	`/admin/modules`
	License	`/admin/license`
	Branding	`/admin/branding`
What it talks to	**Certificates**	`/admin/certificates`
	I/O	`/admin/io`
	Tag drift	`/admin/drift`
	Historian	`/admin/historian`
	Run sources	`/admin/run-sources`
	Linked nodes	`/admin/linked-nodes`
	Fleet	`/admin/fleet`

Group	Screen	Address
	Studio Assistant	`/admin/assistant`

An administrator sees all sixteen screens. Anybody else sees only the screens the node lets them use:

- **I/O** and **Tag drift** for a person the node lets change its I/O;
- **Historian** for a person the node lets configure the historian.

A group with nothing left in it is not drawn, and with no screen at all the menu is not drawn. *Administration in the browser* describes each screen.

The alarm indicator

The indicator beside your name tells somebody who is working in the Studio or on an administration screen that the line is in trouble.



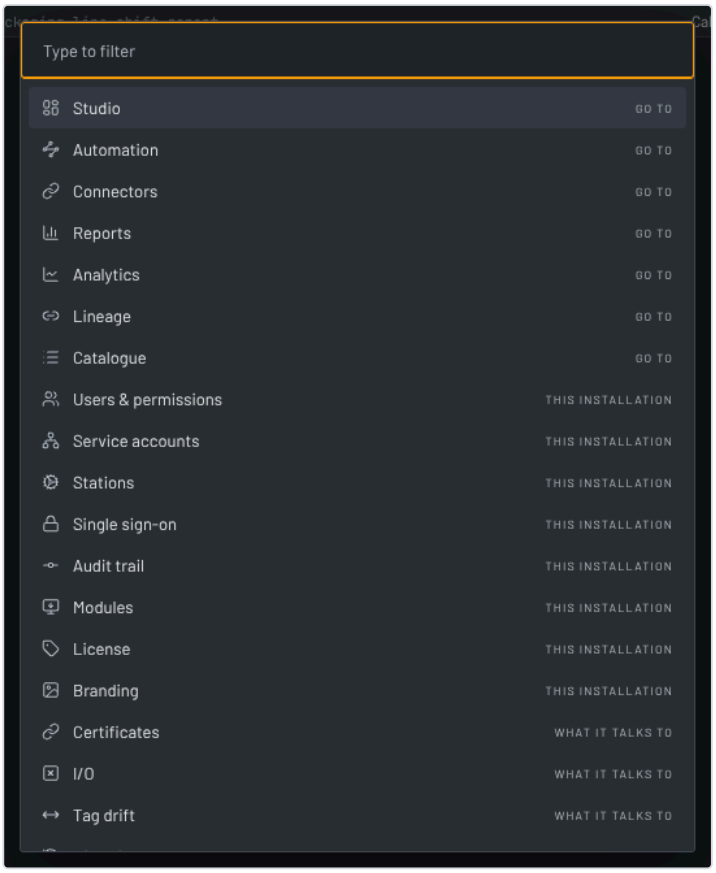
The end of the top bar: the Go to or do... box with Ctrl+K in it, the alarm indicator with a count of one, the name admin, the theme button and Client.

It shows	Meaning
Clear	Nothing is standing at Warning or above
A number	How many alarms (Error and above) or warnings are standing. With both, it shows the alarms and then the warnings, for example `2 · 1`. Its colour is that of the worst
—	The alarm state could not be read. The tooltip says *Alarm state unknown*. It is drawn in the warning colour, because "no alarms" and "could not find out" must never look alike
nothing	This installation has no alarm store to ask, or has not asked yet

Hover over it for the counts in words, for example *Alarms on this installation: 1 Error*. Pressing it opens the operator screens, where the alarm drawer lists the alarms. The indicator covers the whole installation, not one project. It reads the alarm state when the shell opens, and not more often than once every 30 seconds. It is a reminder, not a live feed: look at the operator screens for the alarms themselves.

The command palette

Go to or do… in the top bar opens the command palette. **Ctrl+K** (or **⌘K** on a Mac) opens and closes it from anywhere in the shell, even while nothing has focus and even while you are typing in a field.



The command palette opened with Ctrl+K: the Type to filter box, then Studio, Automation, Connectors, Reports, Analytics, Lineage and Catalogue marked Go to, and the administration screens marked This installation or What it talks to.

The palette is a short list, not a search engine. It holds the work destinations, each marked **Go to**, then the administration screens, each marked with its group (**This installation** or **What it talks to**). It offers exactly what the bar and the **Administration** menu offer you, so it never lists a place you could not open.

Key or action	Does
typing	Keeps the rows whose name or group contains what you typed, in their original order (*Type to filter*). Typing `talks` keeps the whole **What it talks to** group
↓ / ↑	Moves the highlight. It wraps from the last row to the first
Enter	Opens the highlighted row
Escape, or a click outside	Closes the palette. When you opened it from the **Go to or do…** box, focus returns to the box

The filter starts empty every time the palette opens, and the highlight goes back to the top row each time you type. When nothing matches, the palette says *Nothing matches that*.

Example. To check the historian while you are reading a report: press **Ctrl+K**, type `hist` and press **Enter**. The **Historian** screen opens.

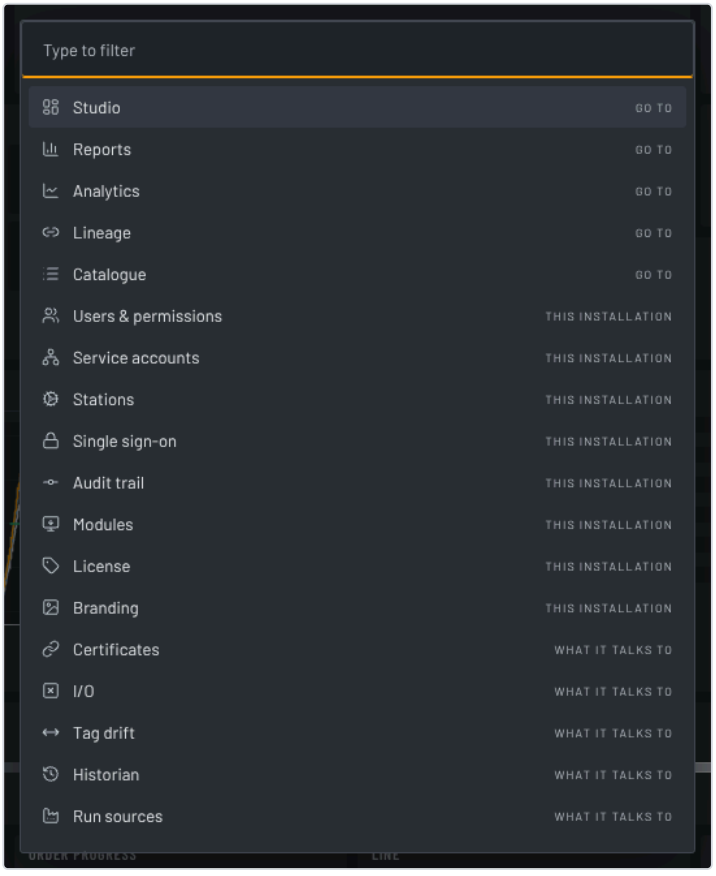
Jumping with Alt+1 to Alt+9

Alt+1 to **Alt+9** open a place by its position: first the destinations in the bar, counted from the left, then the administration screens in menu order. For `admin` on the packaging line, **Alt+1** is **Studio**, **Alt+7** is **Catalogue**, **Alt+8** is **Users & permissions** and **Alt+9** is **Service accounts**. A person with fewer destinations has different numbers, which is why each menu line shows its own.

While the caret is in a field, **Alt+digit** is left to the field, so typing `Alt+1` into a page name does not navigate.

On the operator screens

The palette opens with **Ctrl+K** on the operator screens too. There it lists **Studio** and **Reports** (for those who may open them), **Analytics**, **Lineage**, **Catalogue**, and the administration screens. It does not list **Automation** or **Connectors**. The **Alt+digit** jumps do not work there, so an operator who leans on a key is never taken off the plant view.



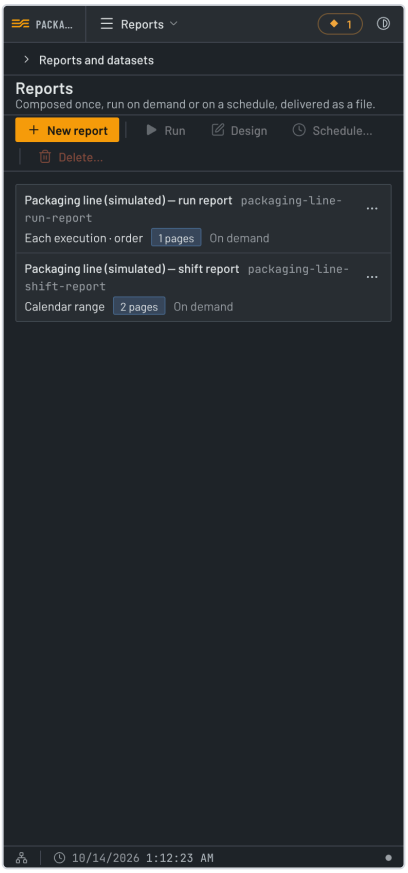
The palette over the operator screens: Studio, Reports, Analytics, Lineage and Catalogue marked Go to, without Automation or Connectors, then the administration screens.

The phone layout

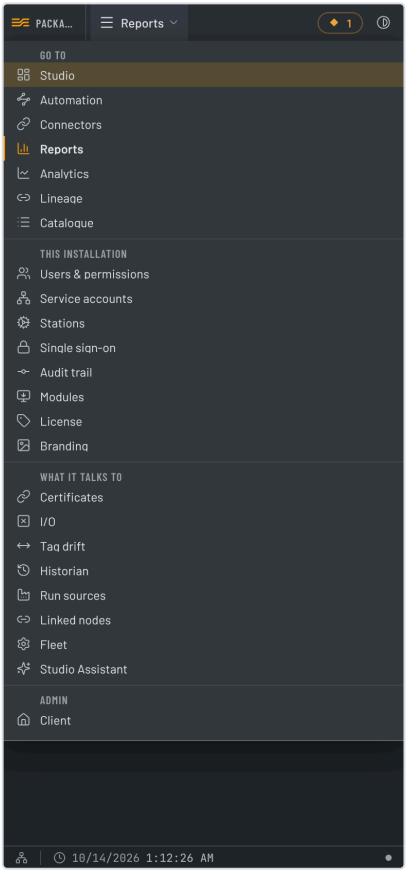
On a screen narrower than 768 pixels the bar keeps only the mark, a shortened site name, the alarm indicator and the theme button. One button, showing the name of the place you are in, opens every destination as a sheet across the window:

Part of the sheet	Holds
Go to	The work destinations, with the current one marked
This installation, **What it talks to**	The administration screens you may open, as in the menu
Foot	Your display name, and **Client**

Press a line to go there. The status bar keeps the time and the Live dot. Below 481 pixels it also drops the site name and the word *Live*.



The Reports screen on a phone: the mark and the shortened site name, the Reports button, the alarm indicator and the theme button in the bar, and the reports as cards; the status bar keeps the time and the Live dot.



The sheet the place button opens on a phone: Go to with Reports marked, This installation and What it talks to with every administration screen, and at its foot the display name admin with Client.

The status bar

The bar at the foot of the shell says where you are and whether the browser is connected.

Part	Shows
Site	The installation's name, beside a connection mark
Date and time	The browser's date and time, updated every second
Where you are (centre)	The trail inside a destination. The Studio's editor fills it with the project and the page you have open. It is empty on screens that are a single page
Screen status (centre)	What the screen below wants to say about itself. Empty on most screens
Live / **Offline**	Whether this browser has a network connection. **Offline** shows a different dot, and its tooltip reads *OFFLINE — displayed values are not live*
Marks	The site's own mark, where Branding supplied one, and the product's mark when the site's mark has taken the top of the shell

The shell's status bar says nothing about the plant: it is drawn on the licence screen too. The operator screens have their own, fuller status bar (live stream, last update, gaps, page version), which *The browser operator HMI* describes.

When a screen stops working

If a screen fails while it is drawing, the shell stays and only that screen is replaced by a panel:

You see	Meaning and what to do
This screen stopped working	The screen below the bar failed. *The rest of the shell is unaffected. Try the screen again, or go somewhere else.*
Try this screen again	Draws the screen again. If it fails again, the panel comes back

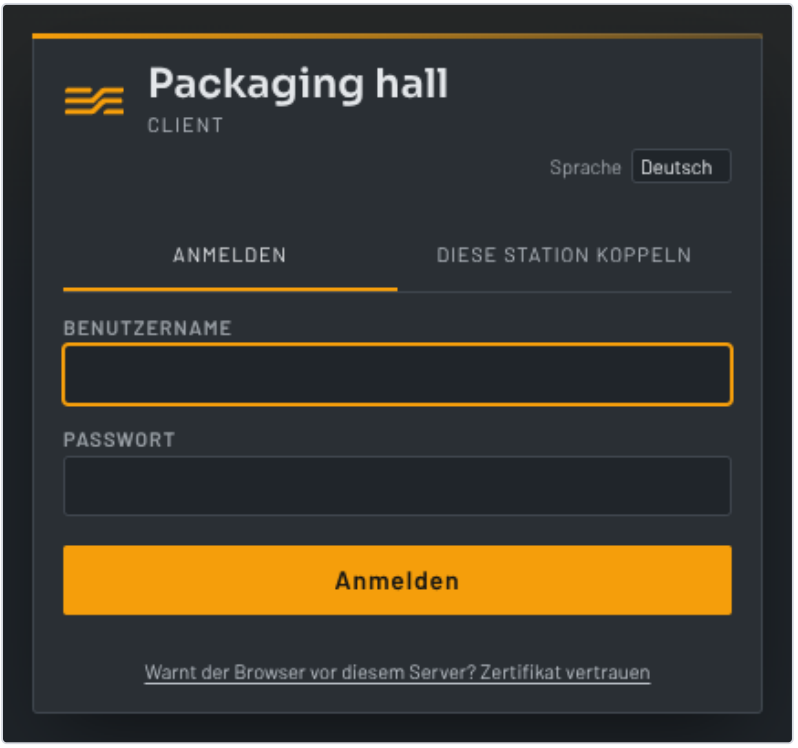
The top bar, the status bar and the palette keep working, so every other destination is still one click away. Going to another destination clears the panel. The operator screens do not use this panel: a mimic that fails there says the plant is not being shown, so an operator never reads a dead screen believing it live.

An address the client does not know shows **Not found**: *That page does not exist in this client*, with **Go to the start page**.

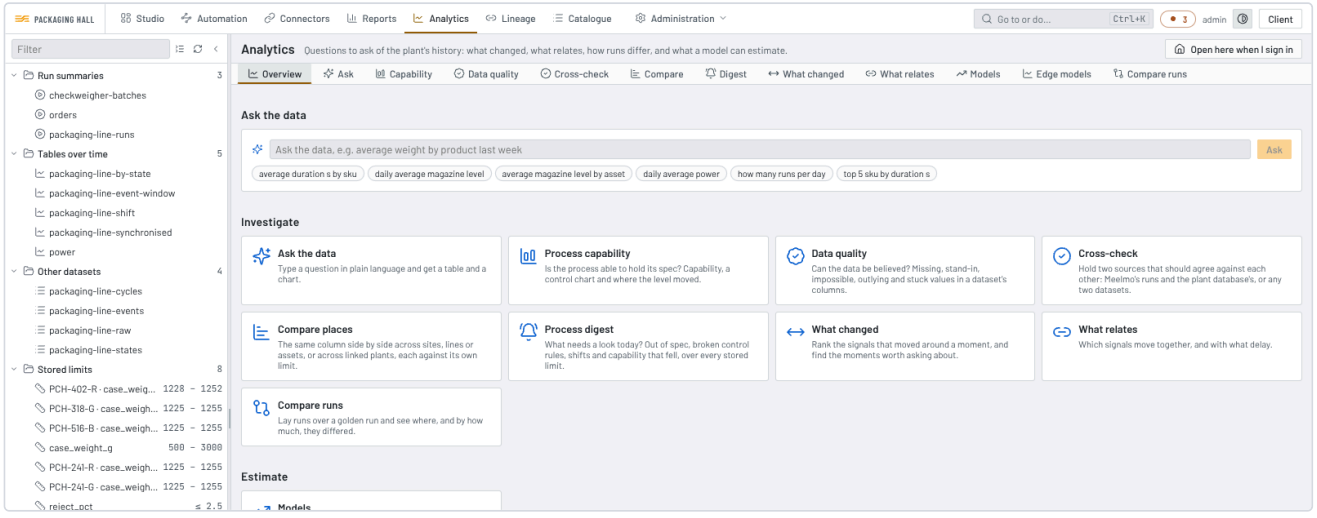
Your own settings

Setting	Where you change it	Where it is kept
Language	**Language** on the sign-in card, or ⋯ ▸ **View** ▸ **Language** on the operator screens. Ten languages are offered, each in its own name	With your session: the choice on the sign-in card carries into the session it opens. A browser you have not chosen in starts from its own language. A paired station opens in the **Language** set for it under **Administration** ▸ **Stations**
Theme	The theme button in the top bar, or ⋯ ▸ **View** ▸ **Theme** on the operator screens. Each press steps through dark, light and, where the site runs another theme of its own, that theme	In this browser. Not offered where the project's theme has **Operators may switch between light and dark** cleared

Setting	Where you change it	Where it is kept
Start page	**Open here when I sign in** on **Analytics**. An administrator can also choose a **Start page** for a person under **Administration ▸ Users & permissions**	With your profile on the server, so it follows you to other browsers. See *Analytics as your start page*
Password	Only when the node requires it: at sign-in, *This account must change its password before continuing* asks for **Current password** and **New password**	At the node



The sign-in card with Deutsch chosen under Sprache: the card itself is now in German — Anmelden, Diese Station koppeln, Benutzername and Passwort.



The Analytics overview after the theme button was pressed once: the same shell in the light theme, with Open here when I sign in at the right of the page title.

Sign out from the operator screens, with **⋯ ▸ Sign out**. The shell has no sign-out of its own, so press **Client** first. A session you leave idle is signed out after the time set under **Administration ▸ Users & permissions ▸ Idle sign-out** (30 minutes unless an administrator changed it).

Example: an engineer who opens Analytics every morning.

1. Sign in, choose **Deutsch** under **Language** on the sign-in card, and sign in.
2. Press **Analytics** in the top bar and click **Open here when I sign in**.
3. Press the theme button once to change from dark to light.
4. Next morning, on another PC, sign in. **Analytics** opens straight away, because the start page is kept with your profile. The theme is dark and the language is that PC's browser language, because those two are kept per browser and per session. Choose **Deutsch** on the sign-in card there too.

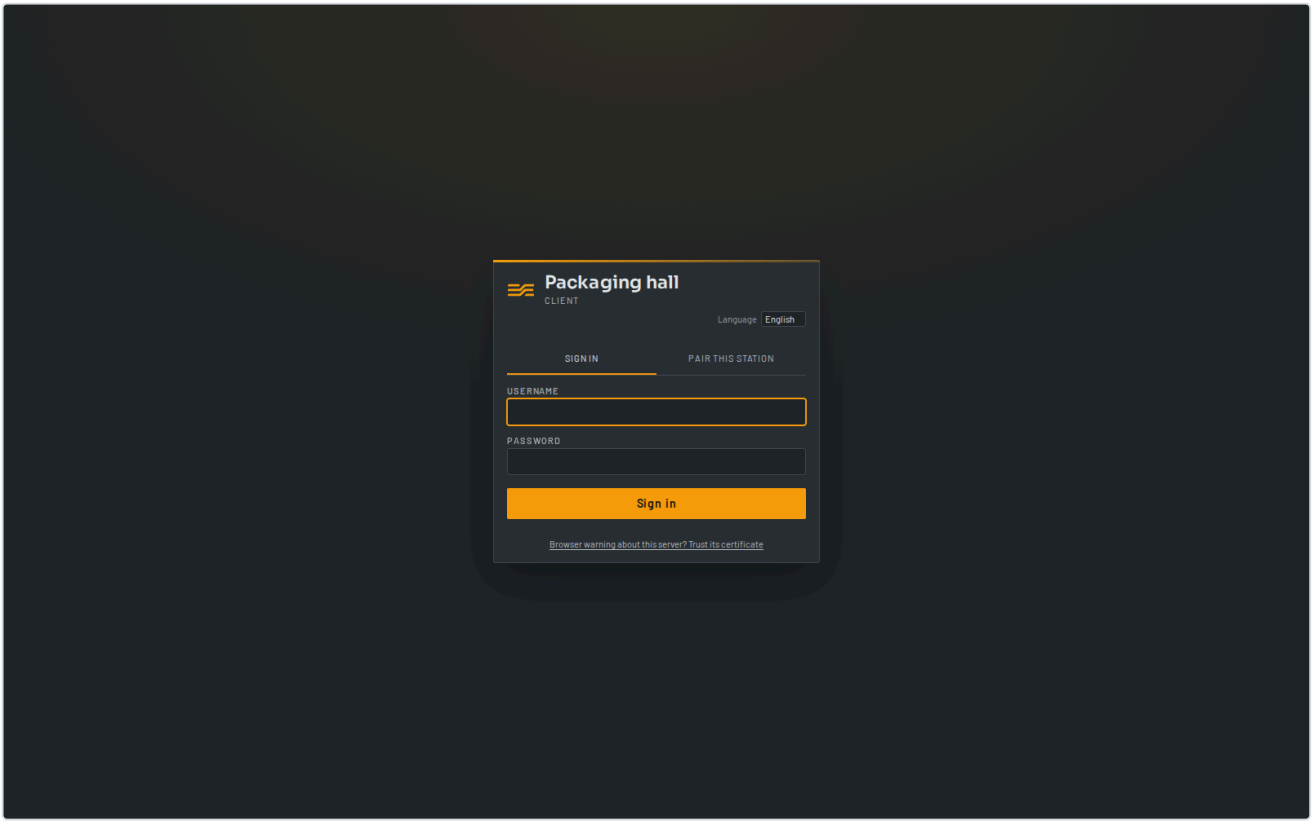
The browser operator HMI

> **Module: Web HMI.** This chapter covers the browser operator screens supplied by the Web HMI module. It is only present on installations licensed for that module. Without it, the live values, trends and alarms described in the base chapters are available in the Desktop client only.

The browser HMI is the operator's surface. It runs in any current browser on a panel PC, a desk PC, a tablet or a phone, can be installed as an app with its own icon, and attaches to a node: every value it shows comes from that node, and every command or acknowledgement it sends is decided by that node.

This chapter walks through what an operator sees and does: signing in, the shell around every page, moving between pages and the kinds of page there are, the demonstration packaging line page by page, the live widgets and trends, cards and dialogs over a page, alarms, history replay, commands, what the screen says when there is nothing to show, stations and phones, the keyboard, and how the site decides which pages each person is shown. How pages are drawn is the subject of *The HMI Studio*; what each tile on a page is, component by component, is in the *HMI component reference*. The top bar, palette and status bar of the other browser screens are described in *The browser client: finding your way*.

Signing in



The browser sign-in panel, with the Sign in and Pair this station tabs, the language selector and the certificate help link

Open the address your administrator gives you, for example `https://plant-server:8443/`. The HMI is always served over **https**.

Field or control	What it is for
Language	The interface language for this browser. Ten are offered; the choice is remembered.

Field or control	What it is for
Sign in tab	Sign in as a person, with **Username** and **Password**.
Pair this station tab	Bind this browser to a station profile with a **Pairing code** an administrator issues under **Administration ▸ Stations**. The code is valid for a few minutes and can be used once. A paired station is served the pages assigned to that station, whoever is signed in.
Sign in button	Checks the name and password against the node's accounts.
Browser warning about this server? Trust its certificate	Opens the certificate trust page (see below).

The username and password are the node's accounts, the same ones the Desktop uses. On the demonstration installation the seeded accounts are `admin` (Administrator) and `shift` (Operator).

What you may see instead of the first page:

Message	Meaning
That username and password do not match.	Wrong name or password. On a node-attached host the message adds that it takes an account the node knows.
Too many attempts. Wait a minute and try again.	Repeated failures are throttled.
This account must change its password before continuing.	Enter **Current password** and **New password**, then **Change password**.
That pairing code is not valid, has expired, or has already been used.	Ask the administrator for a new pairing code.

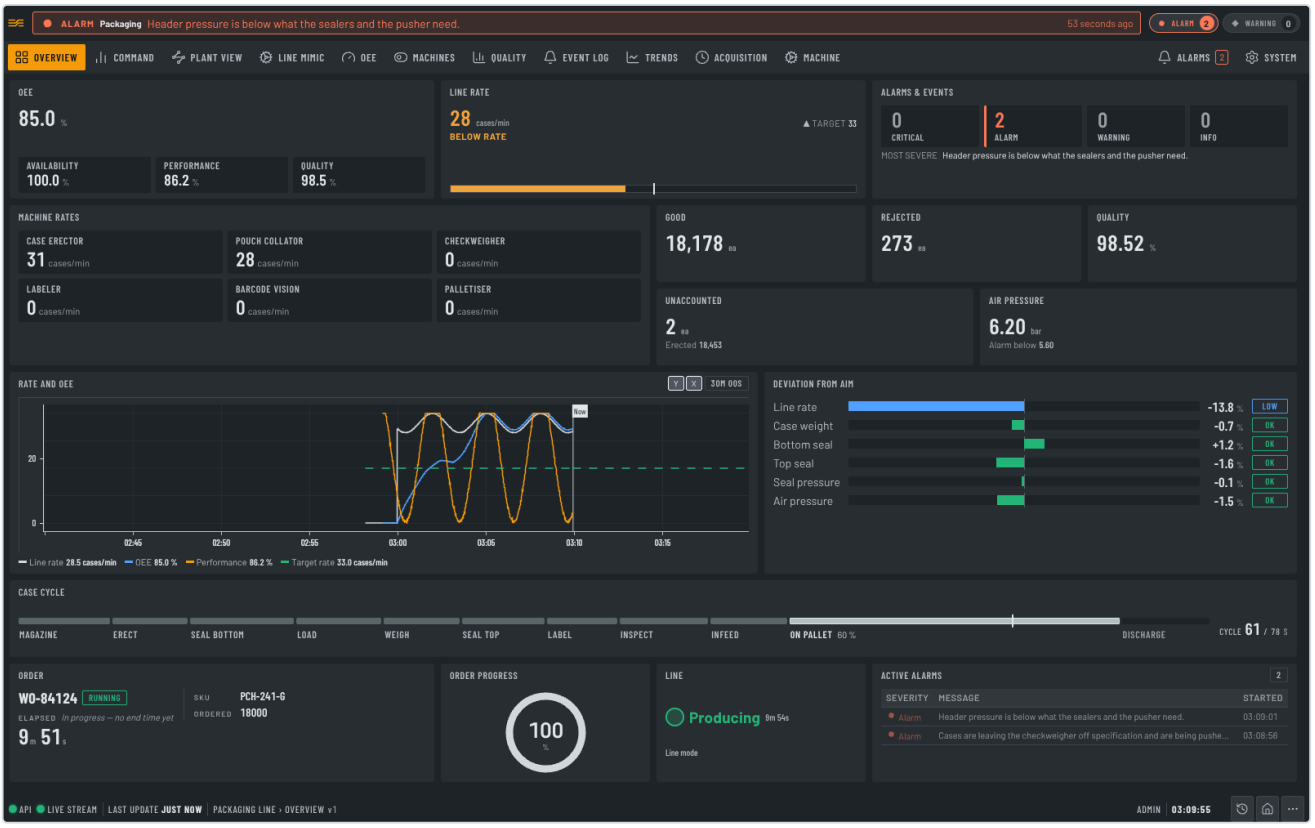
The session is held in a cookie (HttpOnly, Secure, SameSite=Strict), and a session left idle is signed out.

Trusting the site's certificate

A plant server issues its own certificate. A browser that has never seen it reports the site as *Not secure*. Clicking through the warning is **not** enough: the page loads, but the live connection is refused, so every value stays stale and the status bar shows the live stream down.

- **On a Windows machine that Meelmo Setup installed**, nothing needs doing: setup trusted the site's root certificate.
- **On any other device** (tablet, phone, a PC with only a browser), follow the link on the sign-in panel, or open `https://<server>:8443/trust`. The page shows the certificate's fingerprints, to compare with those on the plant server before trusting 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 step (it asks for administrator rights once). Close and reopen the browser afterwards.

The shell: header, page rail and status bar



The Overview page of the packaging line: header with the site and page name and the alarm counts, the page rail, the dashboard tiles and the status bar

Every page is drawn inside the same frame, so an operator always knows where to look.

Area	What it shows
Meelmo mark (top left)	Opens the page rail when the rail is set to overlay, and always on a phone. On a rail down the left or right edge it puts the rail away and brings it back (see *Where the rail sits* below).
Site and page name	The project (here `PACKAGING LINE`) and the page (`Overview`), with *No active alarms* after them. They share the alarm banner's strip: when an alarm stands, the banner takes it over. A green ring beside the site means the plant is clear.
Alarm banner	The worst unacknowledged alarm, with its severity, asset, message and age, for example `ALARM Packaging Compressed air low 14s ago`. When nothing is active it reads *No active alarms*. Pressing it opens the alarm drawer.
Alarm counts	Live counts of **Alarm** and **Warning** (and **Critical** when there is one). They are on every page, so an operator on the OEE page sees an alarm raised elsewhere. Pressing them opens the alarm drawer.
Site selector	Shown only when the viewer is entitled to more than one project. See *The site switcher* below.
Page rail	One tab per page in the project's navigation, each with its icon. Across the top or the bottom of the screen a tab is a slim row — the icon beside the page's name — and the current page's tab is filled in amber; down the left or right edge, and in the phone's drawer, each button is the icon over its name. When the tabs do not all fit, a › button at the end scrolls the rest into view. Where the project groups its pages into areas, across the top or the bottom the rail shows the areas, and a second, lighter bar between it and the page shows the pages of the area you are in — the current one underlined in amber; pressing an area opens its first page. A page outside every area has no second bar. Down the left or right edge and in the phone's drawer, an area's pages are listed under it.

Area	What it shows
Alarms and **System** (right end of the rail)	**Alarms** opens the alarm drawer; **System** opens the system panel.
Status bar (bottom)	**API** and **Live stream** lamps, **Last update** (for example *just now*), **Gaps** with a count when the server has reported data this browser will not receive, the project, page and published version (`PACKAGING LINE › OVERVIEW v1`), the page's next action when its author wrote one (after a ▸), the signed-in user (on a paired station, the station id) and the clock.
Replay, Home, More (bottom right)	The history replay button, **Home** (the start page) and **More** (the menu). When the project turns the footer off, these three move to the right end of the header.

Where the page stops updating, the status bar says so. A green **Live stream** lamp and *Last update just now* mean the values are current. An amber lamp and a growing age mean the page is drawing the last values it received; a value that has gone stale is marked as stale rather than presented as current. A page that shows no live signal at all — the packaging line's **Event log** is alarm and event lists only — has no stream to keep open: its lamp is grey, not amber, and **Last update** reads *No live signals*. That is not a fault; the lists on it are read from the node as before. A **Connection health** card on such a page (the Event log has one) says the same.

A station follows the node's catalogue as well as its screens: when somebody activates a revision that changes the node's signals, the names and units the station shows follow within about ten seconds, without a reload.

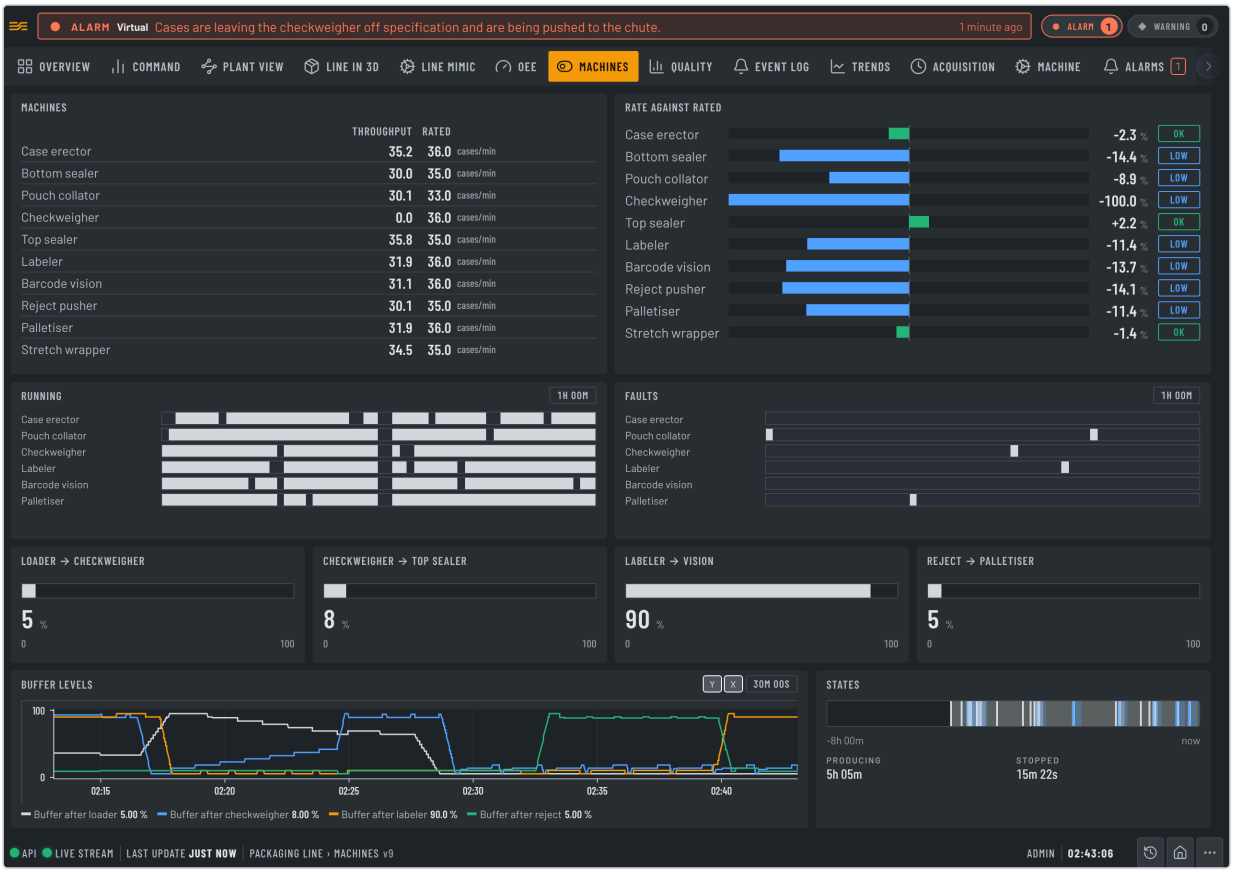
Where the rail sits, and the header

Whoever draws the project chooses where the rail goes and which bands are drawn, in the Studio's **Theme and palette** dialog (see *Theme*). What you see at runtime follows from those choices:

Project setting	What it looks like on your screen
Navigation ▸ Top (the default) or **Bottom**	The rail is a horizontal band of page buttons above (or below) the page. When the buttons do not all fit, a ‹ and a › button appear at the two ends of the band; press them to slide the band along. On a touch screen you can also swipe it.
Navigation ▸ Left or **Right**	The rail is a column of page buttons beside the page. Press the **Meelmo mark** to put the rail away and give its width to the page; press it again to bring it back. This is for the moment only: the next page you open, or a reload, brings the rail back.
Navigation ▸ Overlay	No rail is drawn. Press the **Meelmo mark** (its tooltip reads **Menu**) to open the rail as a drawer over the page; press it again (**Close menu**), or press anywhere on the page behind it, to close it. Choosing a page closes it too.
Show navigation rail cleared	No rail at all. The **Alarms** and **System** buttons live on the rail, so without it the alarm drawer is opened from the alarm banner or the counts.
Show header cleared	The header starts closed. A **Show header** tab sits where it was; press it to open the header on this screen. **More ▸ Hide header** puts it away again. Nothing is changed for anybody else.
Show footer cleared	No status bar. **Replay**, **Home** and **More** move to the right end of the header.
Alarm banner ▸ In header, **Top band**, **Bottom band** or **None**	Where the worst-alarm banner is drawn: in the header, or as its own full-width band above or below the page. **None** draws no banner; the alarm counts stay in the header.

A rail entry can carry entries indented under it; they are drawn as a group of smaller buttons straight after it. A label too long for its button is cut short, and hovering it shows it in full.

A phone always gets the overlay, whatever the project chose: on a screen under 768 CSS pixels wide the rail is the drawer behind the Meelmo mark. The project's setting is not changed, so the same project on a tablet still shows the rail where its author put it.



The Machines page of the packaging line, with the rail running across the top as a row of slim tabs, each page's icon beside its name, and the current page's tab filled in amber

The figure was captured on an earlier build of the demonstration, whose rail also carried a **Machine** entry; today that page opens as a card (see *Machine* below).

The More menu

The ⋯ button at the bottom right (its label is **More**) holds everything an operator sets once rather than uses on every shift. Opening it closes the alarm drawer and the system panel. Its lines are grouped:

Group	Lines	Who sees them
Go to	**Studio**, **Reports**, **Analytics**, **Lineage** and **Catalogue**. **Studio** is offered only to someone who may author pages, and **Reports** only to someone who may view reports. Automation and Connectors are not offered here; they are reached from the other browser screens.	Everybody, filtered by permission.

Group	Lines	Who sees them
Administration (**This installation**, **What it talks to**)	The administration screens, in two groups: **Users & permissions**, **Service accounts**, **Stations**, **Single sign-on**, **Audit trail**, **Modules**, **License**, **Branding**; and **Certificates**, **I/O**, **Tag drift**, **Historian**, **Run sources**, **Linked nodes**, **Fleet**, **Studio Assistant**. See *Administration in the browser*.	An administrator sees all of them. Somebody else sees only the ones the node lets them use: **I/O** and **Tag drift** to someone who may change the I/O, **Historian** to someone who may configure the historian.
View	**Theme** (see below), **Full screen** / **Exit full screen**, **Hide header** (only when the project has turned **Show header** off), **Install** (where the browser offers to install the HMI as an app) and **Language** (a submenu of the ten interface languages, each in its own name, the one in use ticked).	Everybody.
Session	**Sign out**, which opens a one-line submenu, **Sign out?**, as a second step, so a stray tap on a shared operator station cannot end a session in front of a running plant. Closing the menu disarms it.	Everybody.

The menu works from the keyboard like every menu in the product: the arrow keys, **Home** and **End** move through it, typing a letter jumps to the line that starts with it, → opens a submenu and ← leaves it, **Enter** chooses, and **Escape** closes the menu and gives the focus back to **⋯**.

Changing the theme

More ▸ Theme switches this browser between *Industrial dark* and *Industrial light*. When the project's own base theme is something else, for example *Console navy*, it is the third step of the cycle, so you can always get back to it. The choice is remembered by this browser and applies only to it.

Theme is not offered when the project's author has cleared **Operators may switch between light and dark**. That is usual on a wall screen or a control-room panel whose colours were chosen for the room. The page's colours then always follow the project.

The system panel

System at the right of the rail opens a panel for the question "are these numbers current?". It is the panel to read out on a support call when an operator says the numbers look wrong. Close it with its close button or by pressing anywhere outside it. It has six sections.



The System panel over the Overview page: State reads Live with API and Live stream Up and no gaps; Subscriptions lists 15 signals in three groups; then the Session, Project (packaging-demo, page overview, v1), Client and Charts sections, with the Automatic, Hardware and Software switch at the foot

State. Whether this browser is receiving the plant.

Field	What it says
State	The live connection's phase: **Live** when values are arriving; otherwise **Connecting**, **Authenticating**, **Connected**, **Subscribing**, **Reconnecting**, **Disconnected**, **Offline**, **Server unavailable**, or **Session expired** (sign in again).
API	**Up** or **Down**: whether the web host answers requests.
Live stream	**Up** or **Down**: whether the live connection that carries values is open. **No live signals** on a page that shows none, which needs no stream.
Last update	How long ago the last value arrived, **Never**, or **No live signals** on a page that shows none.
Frames	How many frames this browser has received over the live connection.
Gaps	How many gaps the server has reported: stretches of data this browser will never receive. Trends break their lines across a gap rather than joining over it.
Dropped frames	How many frames the server reports it had to drop for this browser.
Reconnects	How many times the live connection has had to be reopened.
Detail	The last refusal or error, in your language, with the server's own English sentence under it. Shown only when there is one.

Subscriptions. What this page asked the server for.

Field	What it says
Signals	How many signals the page (and any card over it) is subscribed to.
Groups	How many subscription groups those signals were planned into.
High frequency	How many are subscribed at the detailed rate (every sample, as a scope receives them).
Trend point budget	The most points a trend on this project is drawn with.
Estimated updates per second	What the page is expected to process.
Values held	How many values this browser is holding.
Stale values	How many of them are stale or not live.

Under the table, one line per group (for example *Fast — 12 signals, …*) opens to list each signal id and its state word: **Live**, **Stale**, **Bad**, **Uncertain**, **No data**, **Not permitted** and so on.

Session. User; **Station** (only on a paired station, its station id); **Roles**; **Composer permissions** (the permissions this session holds); and **Expires** (when the session expires).

Project. Project (the project's id, for example `packaging-demo`), **Assignment** (why this project was chosen for you, from the site's assignments), **Pages** (how many the project has), **Current page** (the page's id, for example `overview`) and **Published version** (for example `v1`).

Client. Client version, **Server version**, **API version**, **HMI schema** and **Minimum client** (the oldest client the server accepts); **Installed as an app**, **Update available** and **Browser online** (**yes** or **no**); and **Viewport**, the window's width and the layout it gets, for example `1920 px — FullHd` (the others are `Phone`, `TabletLandscape`, `Laptop` and `LargeDisplay`).

Charts. How trends are being drawn.

Field	What it says
Renderer	`WebGL` or `WebGL 2` and the graphics driver's name when charts are drawn on the graphics card; **Software canvas —** and the reason (no WebGL, an emulated WebGL, or **chosen**) when they are drawn by the processor; or the SVG fallback when the plot engine could not start.
Hardware canvases	How many charts hold a graphics-card canvas, of how many the browser allows.
Frame budget	The frame rate charts are drawn at, for example `60 fps`.
Device pixel ratio	The screen's pixel density as the browser reports it.

Under it, **Automatic**, **Hardware** and **Software** switch every chart in this browser at once, without losing the samples already drawn. **Hardware** is greyed out where the browser has no WebGL. The switch is not saved: reloading the page returns to **Automatic**. When trends are slow, blank or torn, trying **Software** is the first test.

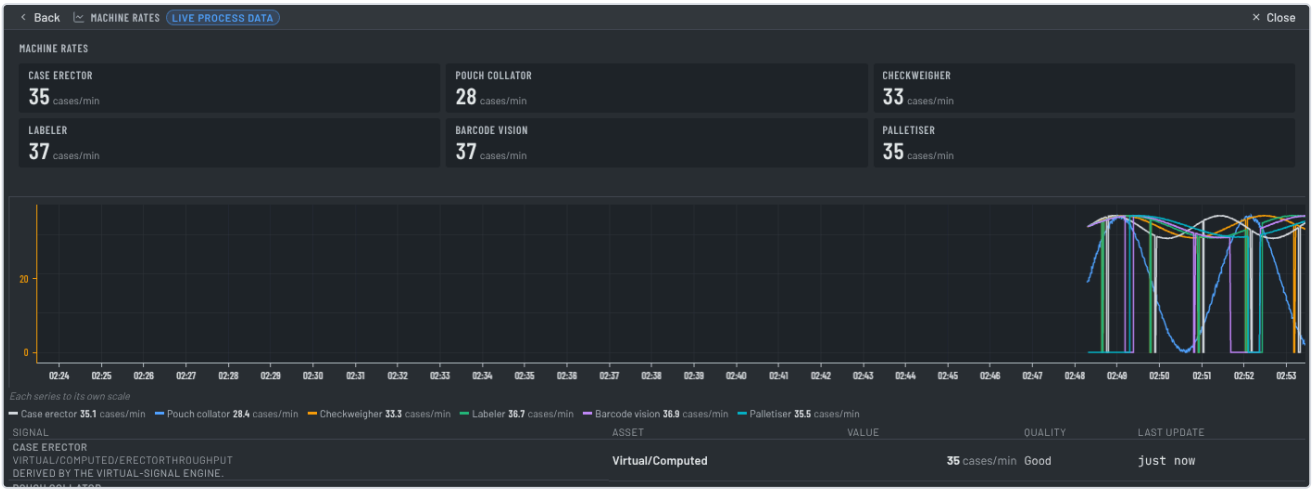
On a support call, read out in this order: **State**, **Live stream** and **Last update**; **Gaps** and **Reconnects**; **Detail** if it is shown; **Stale values**; then **Project**, **Current page** and **Published version**; then **Client version** and **Server version**; and, for a chart problem, the **Renderer** line.

Navigating between pages

Press a page button on the rail to open it. **Home** at the bottom right returns to the start page the site assigned to you (see *Assignments* below). Opening a page closes the alarm drawer, the system panel, any detail overlay and any card standing over the page you leave. The address bar follows: a page's address is `/page/` and its id, for example `https://plant-server:8443/page/oee`, and that address can be bookmarked or opened directly. The browser tab reads the page's name and then the site's name, for example *Line mimic* · followed by the installation's site name, so several tabs side by side can be told apart.

Pages can also link to each other: a tile may open another page, open a small card beside the pointer, or open a dialog over the page; this is set per component by whoever drew the page (see *Cards and dialogs over the page* below).

Many tiles open a **detail overlay** when pressed: a chart of the signal over recent history, a legend with each trace's newest value, and a table of what the catalogue knows about each signal. Pressing one of the signals listed in it opens that signal's own detail; **Back** returns to the tile's. **Escape** steps back once, and closes the overlay when there is nothing to step back to; **Close** closes it at once.



The detail overlay opened from the Checkweigher tile in the Overview page's Machine rates panel: the current rate, the CheckweigherThroughput chart over recent history, its legend with the newest value, and the catalogue row for Virtual/Computed/CheckweigherThroughput with its value, quality Good and last update

On a phone the rail becomes a drawer: press the Meelmo mark to open it and press it again, or anywhere on the page behind, to close it. Pages authored for a desk monitor reflow to two columns on a phone and scroll vertically.

The kinds of page

Every page is one of two layouts, and a 3D model or a page parameter can sit on either. What you can do differs:

Kind	How you recognise it	What you can do on it
Dashboard (grid) page	Tiles in rows and columns, for example **Overview**, **OEE** and **Machines**.	Read and press tiles. The page rearranges itself to the width of the window: each tile has a place for a phone, a tablet, a laptop, a Full HD desk and a large wall display, and where its author drew only some of them the nearest one is used. On a phone the grid is two columns wide and the page scrolls.

Kind	How you recognise it	What you can do on it
Process canvas (freeform) page	A drawing of the plant at a fixed size, for example **Command**, **Plant view** and **Line mimic**, with a zoom control at the bottom right.	The page opens scaled so all of it fits the window, and keeps fitting as the window changes until you zoom. **+** and **−** zoom by steps (the percentage is between them), the mouse wheel zooms, and the fit button returns to the whole drawing. To pan, drag with the middle mouse button or hold **Shift** and drag; an ordinary press is always a press of whatever is under it. The zoom limits are the page's own (on the demonstration pages, 20 % to 600 %).
3D model	A view of the machine drawn from its 3D model, with a row of named part buttons along it, for example **Line in 3D**.	Press a part of the machine: a white outline shows which part is under the pointer, and pressing it opens the detail overlay, headed by the part's name and listing exactly the signals that part carries, over the window its author chose. You stay on the page. Every part that can be pressed is also a button in the row, which is how you reach a part with **Tab** and **Enter**. Where the author ticked **Let the viewer turn it**, drag the model to turn it, use the wheel to zoom it, and double-click the background (not a part) to return to the view the author framed. Otherwise the model stays as it was framed.
One page for many machines	A page that shows *one* machine, chosen by a page parameter, for example **Machine**.	You arrive on it already pointed at a machine, usually from a press on a machine symbol elsewhere. Every value on it comes from that machine.

A 3D model shows its parts in their live state: a part moves only while the signal driving it is good and current, and a part coloured by a threshold says so in a word beside the model (for example `M08_Reject_Pusher` *Rejects high*), because colour is never the only signal. When the model cannot be loaded, the component says **Model unavailable** with the reason, rather than drawing an empty space.

Worked example. On **Line in 3D**, the reject rate has risen above 1.6 %: the reject pusher is painted in the warning colour and *Rejects high* is written beside the model. They press the **Reject pusher** button in the row of part buttons (or the pusher itself): the detail overlay opens, headed *Reject pusher*, with the pusher's signals over the last hour. **Escape** closes it, and the 3D page is still there underneath.

Page parameters: selectors, tabs and the breadcrumb

A page parameter is a value a page reads to decide *what* it shows: which area, which asset, which machine, which time window, which view. The same page then serves many machines or lines. Some components on a page set one when you press them:

Component	What it looks like	What pressing it does
Area selector	A row of buttons, one per area its author listed; the one in force is highlighted.	Sets the `area` parameter, so the page shows that line or area.
Asset selector	A row of buttons, one per asset its author listed.	Sets the `asset` parameter, so the page shows that asset.
Severity filter	**All**, then one button per severity.	Shows only the alarms and events of that severity in the page's alarm and event components. **All** shows them all again.
Asset filter	**All**, then one button per asset its author listed.	Shows only the alarms and events on that asset or under it in the page's alarm and event components.
Time range selector	A row of window buttons, by default `15m`, `1h`, `8h` and `24h`.	Sets the window every trend and chart on the page reads, except a chart its author pinned to its own window.

Component	What it looks like	What pressing it does
Tabs	A row of tabs.	Sets a page parameter (by default named `view`); the components belonging to each tab are shown only while it is selected.
Asset breadcrumb	The asset's path, with one button per level.	Pressing a level sets the `asset` parameter to that level, which is the way back up the plant.

A selector or tabs whose author listed no choices says *No choices configured* or *No tabs configured*. When you choose, the page's values are fetched again for the new choice, so a chart may briefly show nothing while its history arrives.

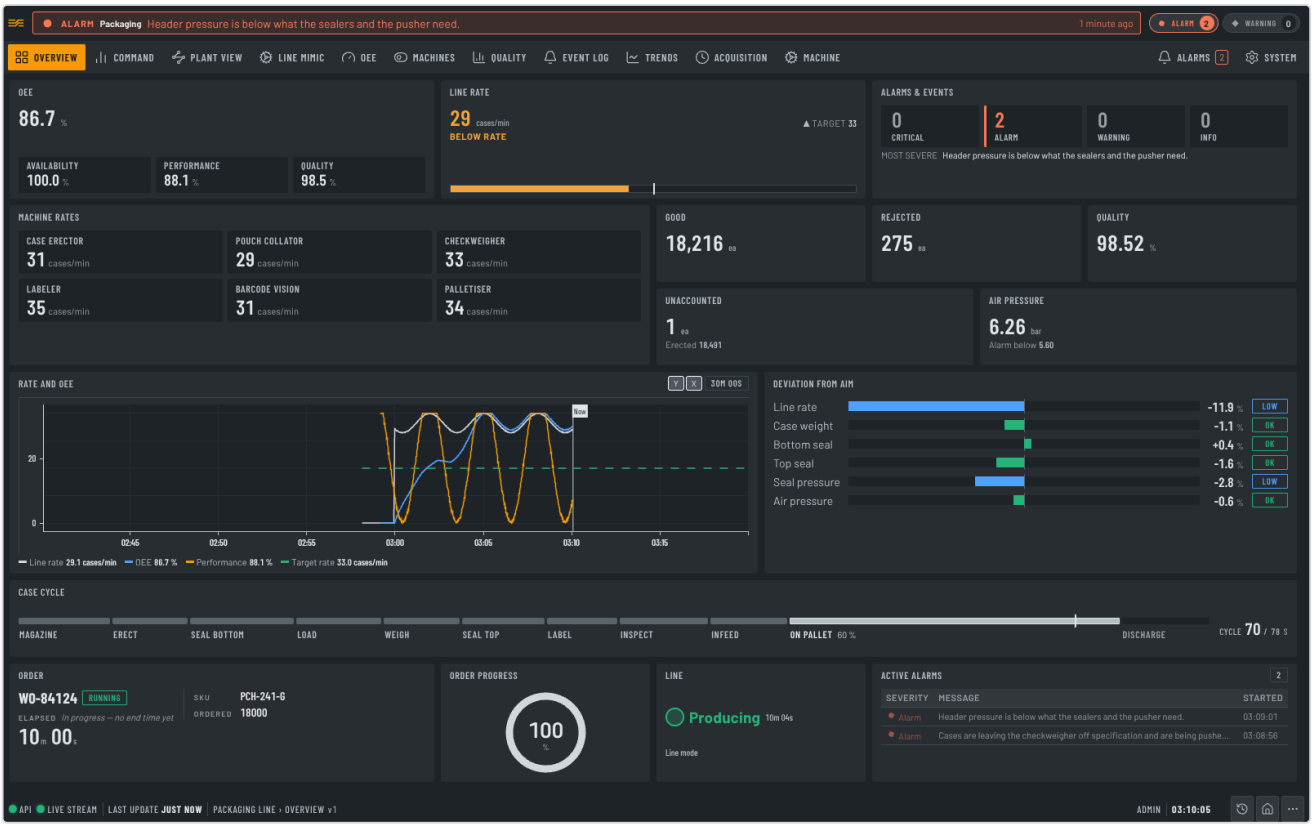
The site switcher

One web host can serve several HMI projects, for example one per line. Which project a viewer gets is decided by the site's assignments (see *Assignments: who sees which pages*). When a viewer is entitled to more than one project, a **Site** selector appears in the header listing them; choosing one switches the whole shell, rail and all, to that project. An administrator sees every project. On a site with one project the selector is not drawn.

The demonstration packaging line, page by page

The demonstration installation ships an HMI project called **Packaging line** (`packaging-demo`) for a simulated case-packing line: case erector, bottom sealer, pouch collator (loader), checkweigher, top sealer, labeler, barcode vision, reject pusher, palletiser and stretch wrapper. Its rail carries **Overview**, **Command**, **Plant view**, **Line in 3D**, **Line mimic**, **OEE**, **Machines**, **Quality**, **Event log**, **Trends** and **Acquisition**. A twelfth page, **Machine**, is not on the rail: it opens as a card when you press a machine on **Line mimic**. The pages below are what an operator sees on it. Some figures were captured on an earlier build whose rail also carried **Machine** and lacked **Line in 3D**.

Overview



The Overview page: OEE, line rate against target, alarm summary, machine rates as a panel of tiles, counts, the rate and OEE trend with its target carried past now, deviation from aim, the case cycle, and at the foot the current order with its progress, the line state and the active alarms

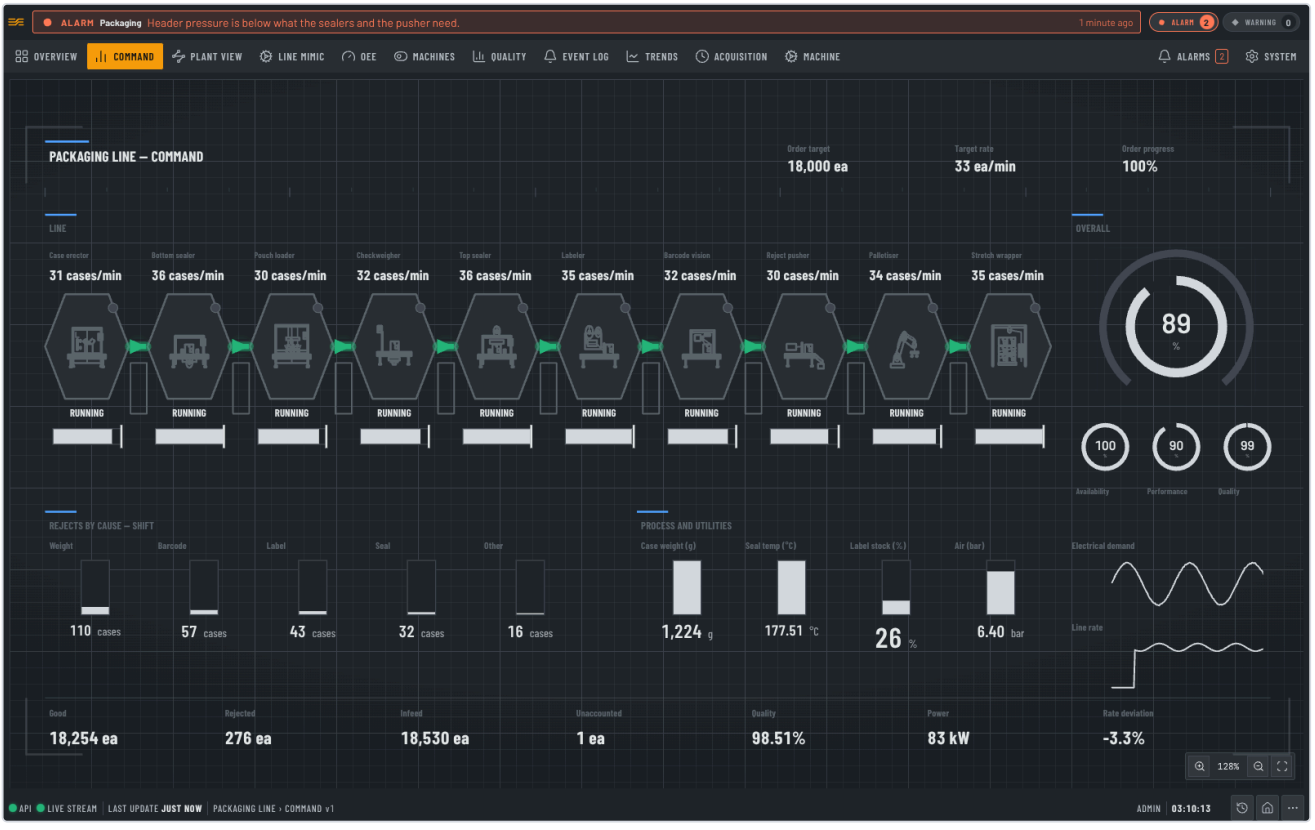
The Overview is the first page most viewers land on. It is a dashboard grid:

Tile	What it shows
OEE	Overall equipment effectiveness, `Virtual/Computed/OEE`, for example `98.1 %`, with its three factors — **Availability**, **Performance** and **Quality** — as tiles under it.
Line rate	Loader throughput against the target rate `Packaging/Line/TargetRate`. The band label **BELOW RATE** and the bar with its target mark (`▲ TARGET 33`) say whether the line is keeping up.
Alarms & events	A severity summary: **Critical**, **Alarm**, **Warning** and **Info** counts. A count of zero is grey; a count that is not is in its severity's colour, with a bar down its edge.
Machine rates	One panel holding a tile per machine — **Case erector** … **Palletiser** — each with its throughput in cases/min.
Good, **Rejected**, **Quality**	Shift counts `Packaging/Line/GoodCount`, `Packaging/Line/RejectCount` and the quality percentage.
Unaccounted	Units that entered and neither left as good nor were rejected, with a caption line under it: **Erected** and the infeed count it is measured against.
Air pressure	`Packaging/Line/AirPressure`, for example `6.51 bar`, with the caption **Alarm below 5.60**.
Rate and OEE	The loader's throughput, OEE and performance over 30 minutes, and the target rate as a dashed line carried ten minutes past the **Now** line.
Deviation from aim	Six readings against their aims as a bar either side of zero, each with its signed deviation in per cent and **OK**, **HIGH** or **LOW**.

Tile	What it shows
Case cycle	The case's eleven steps end to end — the room their names leave is shared half evenly and half by each step's share of the 78-second cycle, so the names read evenly and **On pallet**, 60 % of the cycle, is still the long one — the current one outlined with a line where the cycle has reached, and **Cycle 41 / 78 s** at the right.
Active alarms	The alarm list, at the foot of the page beside the order and the line state: severity, message and start time — the asset and the duration are on the **Event log**. A message too long for its column ends in an ellipsis; hover it for the whole text. *No active alarms* on its first line, with a green dot, when clear.
Order	The current execution: order `WO-84122`, **RUNNING**, elapsed time, and beside them SKU `PCH-402-R` and quantity ordered `10800`.
Order progress	Beside the order, a radial indicator of how much of it is done (`90 %`).
Line	The line's current state, here **Producing** with how long it has held it.

Worked example. At the start of a shift the operator opens **Overview** and reads, top left to bottom right: OEE 98 %, line rate 33 cases/min against a target of 33 (the **BELOW RATE** band is shown while the rate is under target), order WO-84122 at 90 %, no alarms, air at 6.5 bar. Nothing needs attention; the operator moves to **Command** to watch the line.

Command



The Command page: the machine chain with each machine's rate and state, overall and A/P/Q gauges, rejects by cause, process and utility readings, and the shift counters

Command is a freeform process page (a mimic) laid out as a control-room screen:

- Across the top: **Order target** (`10,800 ea`), **Target rate** (`33 ea/min`) and **Order progress** (`91%`).

- **LINE**: the ten machines as hexagonal symbols joined by flow arrows. Above each, its rate in cases/min; below each, its state word (**RUNNING**, **BLOCKED**, **FAULTED**, **STARVED**) and a bar of rate against rated speed. The symbol's outline takes the state's colour (amber for blocked or starved, red for faulted), and the word is always written, so the state never depends on colour alone. The arrows between machines turn green while material flows.
- **OVERALL**: a large OEE gauge (`96 %`) and three small gauges for **Availability**, **Performance** and **Quality**.
- **REJECTS BY CAUSE — SHIFT**: vertical bars for **Weight**, **Barcode**, **Label**, **Seal** and **Other** with the case counts.
- **PROCESS AND UTILITIES**: **Case weight (g)** (for example `1,255 g`), **Seal temp (°C)**, **Label stock (%)**, **Air (bar)**, and mini trends of **Electrical demand** and **Line rate**.
- The bottom row: **Good**, **Rejected**, **Infeed**, **Unaccounted**, **Quality**, **Power** and **Rate deviation**.

The zoom control at the bottom right of a process page (**+**, the zoom percentage, **−**, and fit to screen) scales the drawing; the mimic keeps its proportions.

Order target and **Target rate** on this page are the two command controls in the demonstration project: each is a `WriteTag` interaction on `Packaging/Line/OrderTarget` and `Packaging/Line/TargetRate`, with confirmation required. See *Commands* below for what has to be true before pressing one does anything.

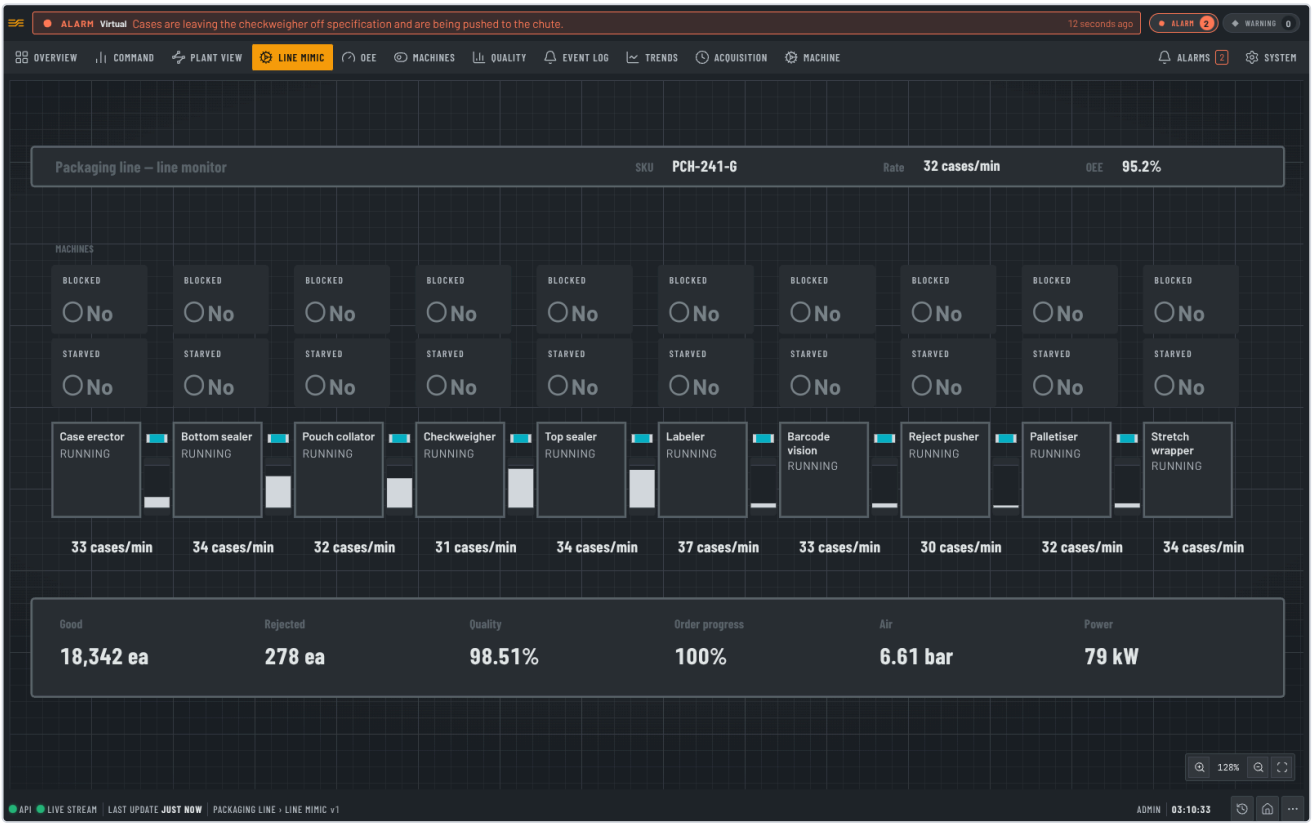
Plant view



The Plant view page: the case-packing line drawn as equipment symbols with each machine's rate and state, OEE, line rate and order, and the shift-to-date counters

Plant view is a second mimic of the same line, drawn with equipment symbols that look like the machines (erector, sealers, collator, checkweigher, labeler, vision, pusher, robot palletiser and wrapper). Each carries its rate above and its state below. At the top right are **OEE**, **Line rate** and **Order**; **SHIFT TO DATE** along the bottom gives **Good**, **Rejected**, **Unaccounted**, **Quality**, **Air** and **Power**. It is the page to put on a wall screen visible from the line.

Line mimic



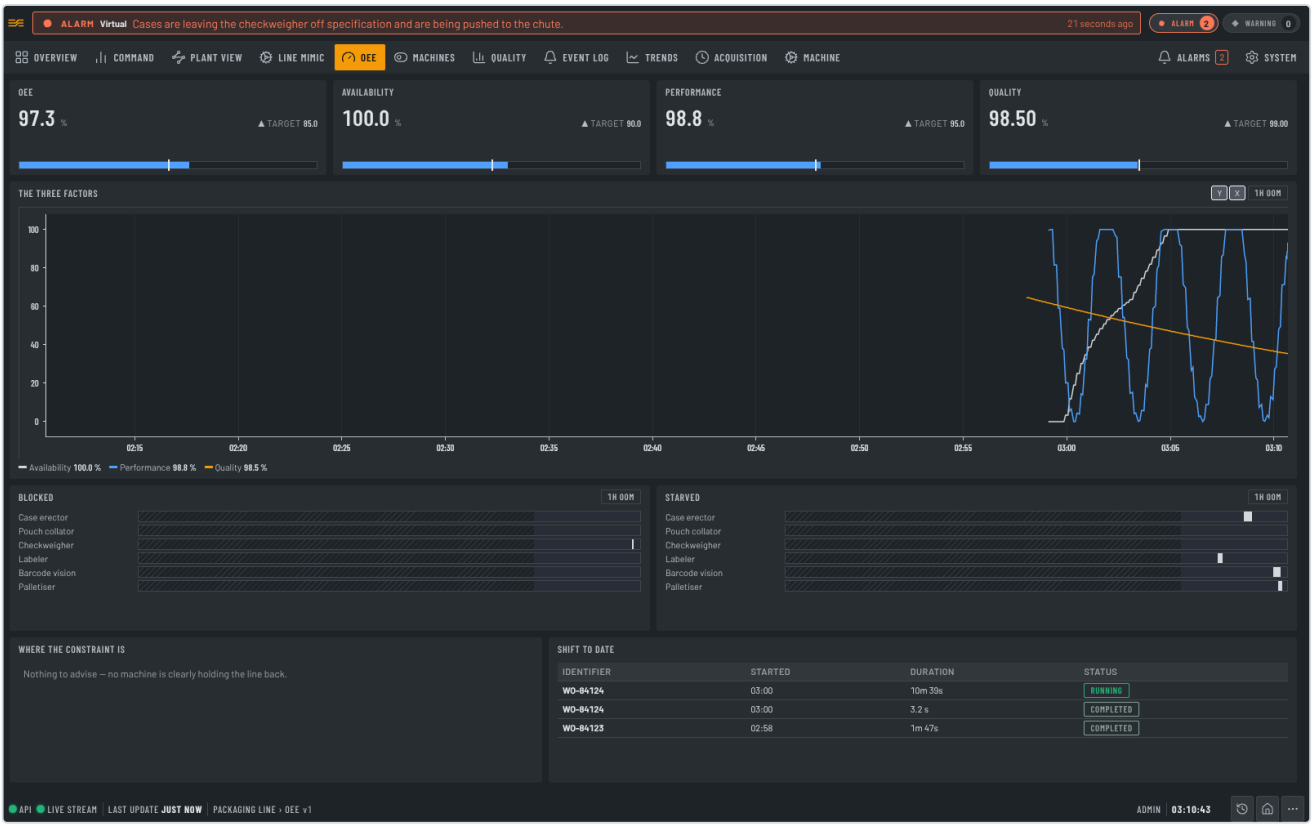
The Line mimic page: a line monitor strip with SKU, rate and OEE, a blocked and starved lamp for each machine, machine boxes with running state and buffer bars, and the summary counters

Line mimic is the line monitor used to find *where* the line is losing time:

- A header strip: **SKU** `PCH-402-R`, **Rate** `30 cases/min`, **OEE** `89.6%`.
- **MACHINES**: for each of the ten machines, a **BLOCKED** and a **STARVED** lamp (**No** with an empty ring when false), a box with the machine name and **RUNNING** or its state, and a vertical buffer bar showing how full the conveyor to the next machine is.
- The machine rates along the bottom, and a summary of **Good**, **Rejected**, **Quality**, **Order progress**, **Air** and **Power**. The **?** beside **Rejected** opens an information panel.
- Pressing a machine box opens the **Machine** page for that machine as a card beside the pointer (see *Cards and dialogs over the page*).

Worked example. The labeler's **STARVED** lamp lights while the checkweigher's buffer bar is empty: the labeler is waiting for cases the checkweigher is not sending. The operator reads the checkweigher box and finds it **RUNNING** but slow; the cause is upstream of the labeler, not the labeler itself.

OEE



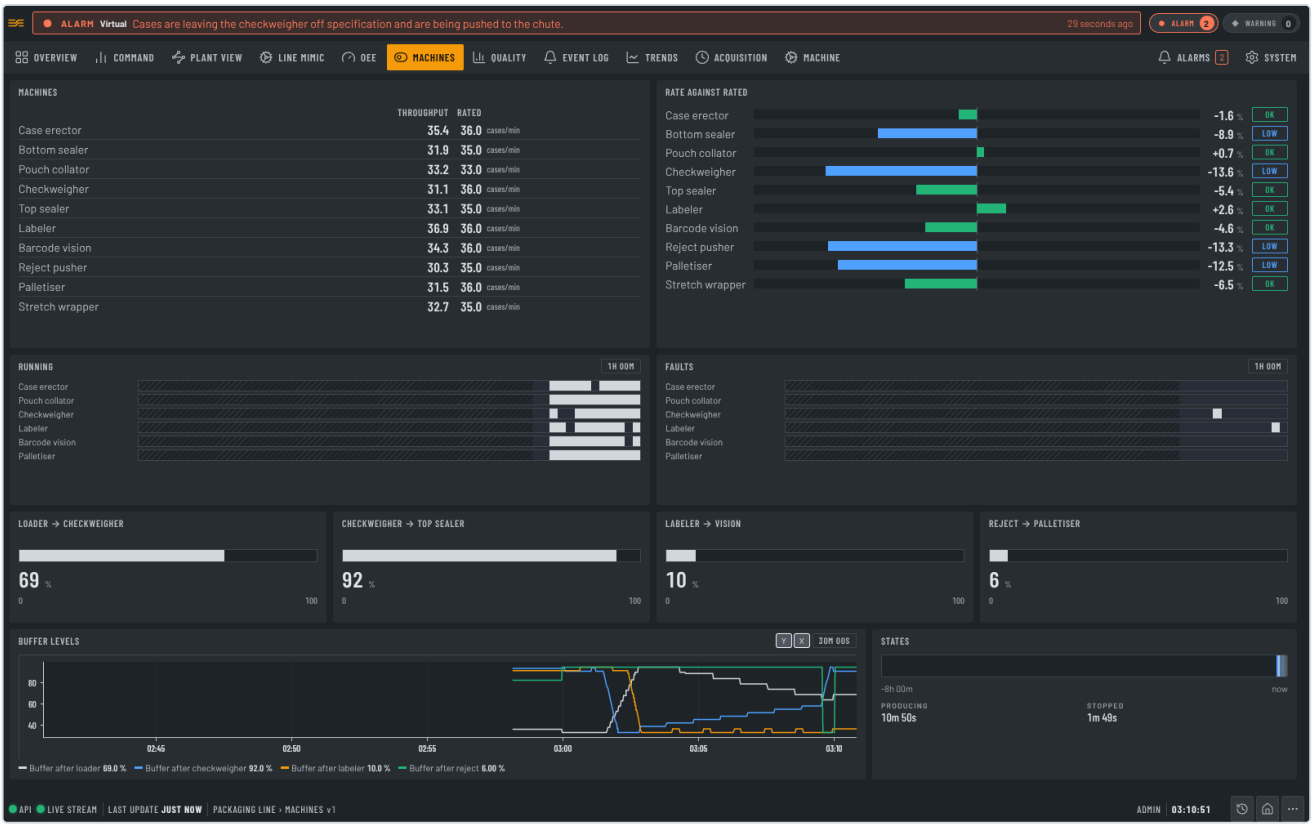
The OEE page: OEE, availability, performance and quality cards, a trend of the three factors, blocked and starved timelines per machine, the constraint advisory and the shift-to-date table

OEE breaks the headline figure into its three factors:

- **OEE**, **Availability**, **Performance** and **Quality** as large figures (here `87.2 %`, `100.0 %`, `88.4 %`, `98.66 %`), each against the level a line held to world class reaches — ▲ **TARGET** 85, 90, 95 and 99 % — with a bar to it at the foot of the tile.
- **THE THREE FACTORS**: a one-hour trend of availability, performance and quality.
- **BLOCKED** and **STARVED**: a boolean track per machine over the last hour. Each machine's row fills where it was blocked (or starved), so the pattern of who waited on whom is visible at a glance. Where the node recorded nothing — before it started, or while it was stopped — the track is hatched rather than drawn as *not blocked*.
- **WHERE THE CONSTRAINT IS**: the advisory panel. It names the machine holding the line back when one is, with the margin and the window; here it reads *Nothing to advise — no machine is clearly holding the line back.*
- **SHIFT TO DATE**: the runs in the shift, with **Identifier**, **Started**, **Duration** and **Status** (`WO-84122`, `01:51`, `4m 51s`, **RUNNING**).

An advisory is a sentence, not a command: it is there to be read by somebody who knows the line.

Machines



The Machines page: a table of machine rates against their ratings, the rate-against-rated deviation panel, running and fault timelines per machine, buffer level gauges between machines, a buffer level trend and the line state timeline

Machines is the dashboard for the machines as a group:

- **MACHINES**: a table of the ten machines, a row each, with **Throughput** and **Rated** in columns and the unit at the end of the row.
- **RATE AGAINST RATED**: each machine's throughput as a deviation from its rated speed, in per cent, with **OK** within 8 % and **LOW** or **HIGH** beyond it — a machine that has stopped reads **LOW**.
- **RUNNING**: a one-hour boolean track per machine.
- Four buffer gauges: **Loader → Checkweigher**, **Checkweigher → Top sealer**, **Labeler → Vision** and **Reject → Palletiser**, each a bar from 0 to 100 % with the current fill (for example `90 %` between the labeler and vision, which means vision is holding cases back).
- **FAULTS**: a one-hour boolean track per machine.
- **BUFFER LEVELS**: a 30-minute trend of the four buffers.
- **STATES**: the line's state timeline over the last eight hours, with the current state (**PRODUCING**, `5m 05s`) below.

Quality, Event log, Trends, Acquisition, Line in 3D and Machine

The remaining pages of the demonstration project follow the same pattern:

Page	What is on it
Quality	Rejects by cause (bar chart), yield, rejected count, case weight against target (`Packaging/Checkweigher/CaseWeight` against `Packaging/Checkweigher/WeightTarget`), a case weight distribution, a trend of the bottom seal temperature, weight statistics, the weight's deviation with its target under it, and a label stock gauge. The causes and statistics are named in words (**Weight**, **Case weight**) rather than by tag.
Event log	The active alarm banner, a severity counter, the alarm list, the event list and an alarm detail panel. See *Alarms* below.

Page	What is on it
Trends	A live scope, a data logger, a stacked trend, a multiplot, and the case weight as a point per case against its specification carried past now. See *Live trends* below.
Acquisition	Connection health and trends of requested against achieved acquisition period, jitter and dropped samples: the page to open when values look late.
Line in 3D	The case-packing cell drawn from its 3D model, with a row of part buttons (**Case erector** … **Stretch wrapper**) and, beneath it, order progress, the robot's state, the pallet count, cycle time, cases per hour and reject rate. See *The kinds of page* above.
Machine	One page that serves every machine through the `machine` page parameter: state with time in state, rate against rated speed, a 30-minute mini trend of the rate and a state track, for the machine chosen. It is not on the rail; it opens as a card from **Line mimic** (see *Cards and dialogs over the page*).

The live widgets: value, gauge, state and mimic

Every page is built from the component types of one catalogue, shared by the server that validates pages and the browser that draws them. Four kinds carry most live information.

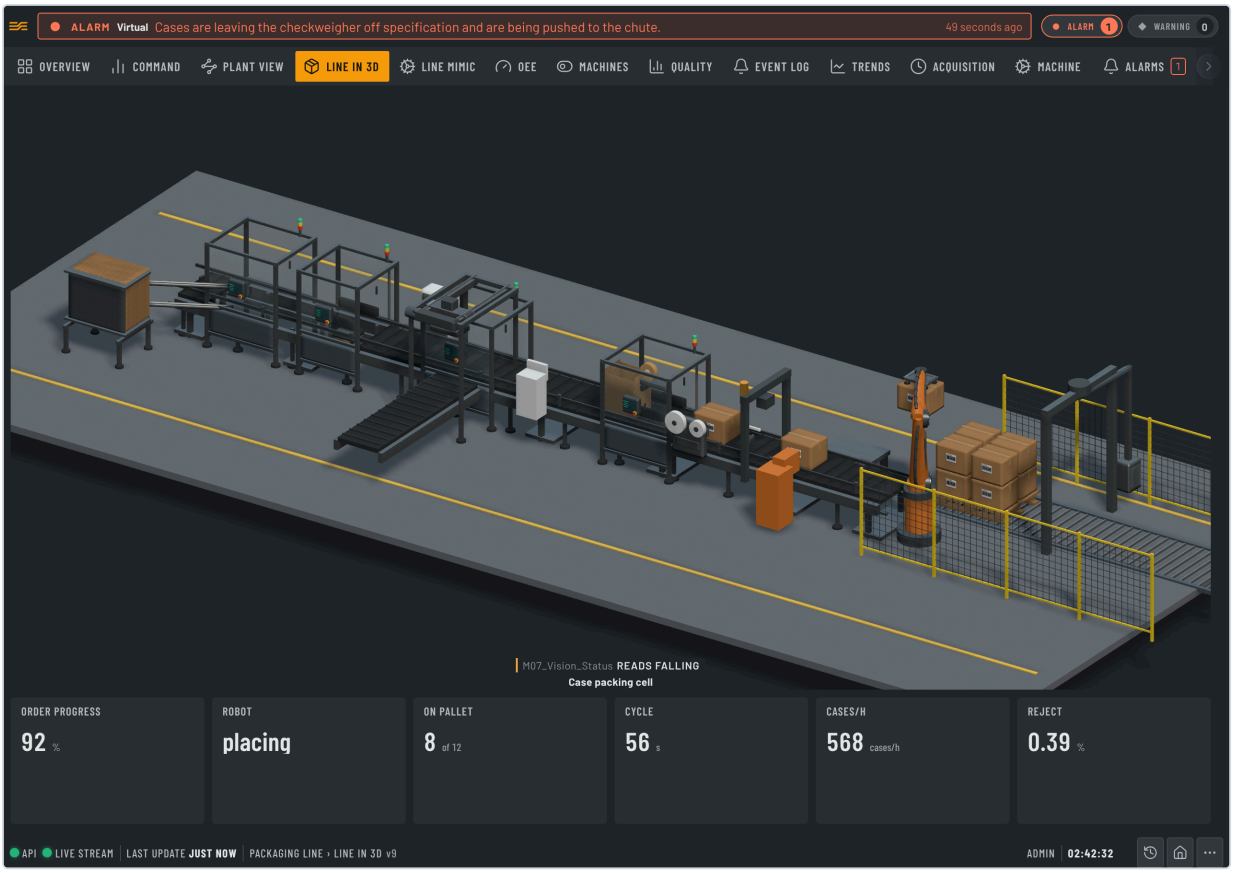
Widget	What the operator reads	Example on the packaging line
Value (numeric value, KPI card, counter, percentage, rate card)	A number with its unit, optionally a mini trend, a target and a band label.	**Air pressure** `6.51 bar`; **Good** `16,282 ea`.
Gauge (linear, vertical, radial progress, range, segmented)	A value against a scale, often with a target mark.	The buffer bars on **Machines**; the OEE gauges on **Command**.
State (state indicator, traffic light, current state)	A discrete state as a word and a colour.	**Producing** on the **Overview** line tile; **RUNNING** / **BLOCKED** on **Command**.
Mimic (process symbols and shapes on a freeform page)	The plant drawn as itself, with shapes animated from signals.	The machine chain on **Command** and **Plant view**.

Some conventions hold on every widget:

- **Colour never works alone.** A value in a threshold band shows the band's label (for example **BELOW RATE**) as well as its colour.
- **Quality and staleness are shown.** A value with bad quality or no recent update is marked, not drawn as if it were current.
- **Pressing opens detail.** Most value tiles open the detail overlay; some open another page or a card, as the page author decided.

The four kinds above are a summary. The catalogue has about a hundred component types, from counters and gauges to batch, run and relationship panels; what each one shows, what its states and colours mean and what pressing it does is listed type by type in the *HMI component reference*.

Where a site has a 3D model of its line, a model view is an ordinary page component: pressing a part of the machine opens its detail, headed by the part's name, listing exactly the signals that part carries. Every pressable part also appears as a named button along the top of the model, so it can be reached from a keyboard.



A 3D view of the case-packing line with the cell's order progress, robot state, pallet count, cycle time, cases per hour and reject rate beneath it

Live trends: scope, logger, multiplot and stacked



The Trends page with a live scope, a data logger of machine rates since the page opened, a stacked trend, a multiplot, and the case weight as a point per case against its specification carried past now

Several trend components exist, and the difference matters:

Widget	What it is for
Trend (live trend, historical trend, multi-signal trend)	The plant's recent history, updating at an interval.
Scope	A live waveform: every acquired sample, not a decimated snapshot. Shows a short window, for example 30 s.
Logger	An accumulating record that grows from the moment the page opened, for example **Machine rates — since open**.
Multiplot	Several signals on one time axis, sharing one plot area.
Stacked	Several signals on one time axis, each in its own band with its own vertical scale.

A trend samples what the server sends at an interval; a scope and a logger receive every acquired sample over a lossless transport. 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. When two components ask for the same signal at different rates, the fastest wins.

Controls in a trend's header:

Control	What it does
Y	Shows or hides the value axis. Hiding it gives the plot more room when you want the shape rather than the numbers.
X	Shows or hides the time axis.

Control	What it does
Window chip (for example **30M 00S**, **1H 00M**)	The time span shown.

The axis toggles are a view choice for the person reading: reopening the page gives back what the page author chose. On a trend that offers it, dragging across the chart replays just that span (*Drag across the chart to replay just that span*).

The legend under a trend names each series and its latest value with unit (`LoaderThroughput 33.5 cases/min`). When the legend's tooltip reads *Each series to its own scale*, each trace is scaled independently: the scale down the side, when the **Y** toggle shows it, is the first series' own, in its colour; read the others' values from the legend.

Cards and dialogs over the page

A component can open a whole page *over* the one you are on, in one of two forms, chosen by whoever drew it.

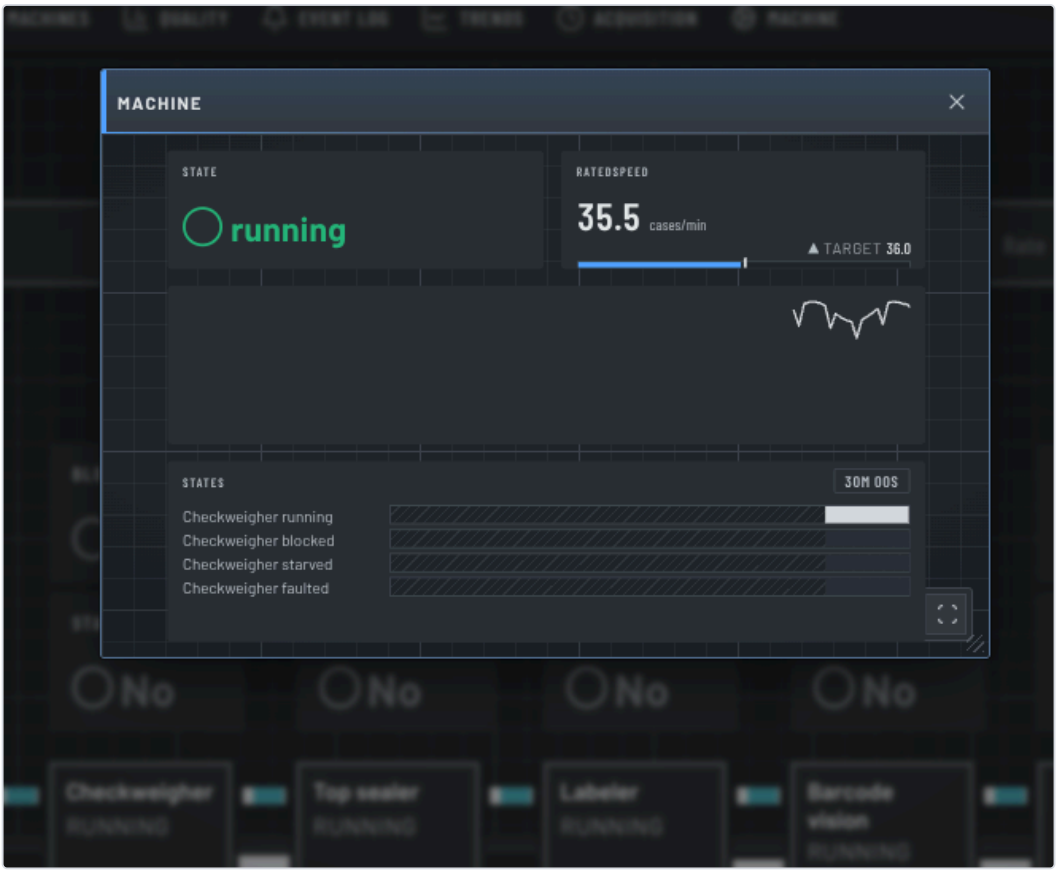
	Card	Dialog
Where it opens	Beside the pointer, just below and to the right of what you pressed, pulled back inside the window when there is no room below or beside it.	In the middle of the screen.
The page behind	Shaded but still live: its values keep updating.	Dimmed.
Size	The size of the page it shows, within limits, unless its author set one.	The same.
Closing it	**Close** (the × in its title bar), **Escape**, or a press anywhere on the page behind it.	**Close**, **Escape**, or a press on the dimmed page behind it.

While a card or dialog is open it is the one thing you can act on: a press on the page behind closes it rather than pressing whatever is underneath, so a stray touch on a mimic cannot operate anything. Only one dialog is open at a time.

Both have a title bar naming what they show: the page's name, or the tile or signal being looked at inside it.

- **Move it** by dragging its title bar (its tooltip reads *Drag to move*), to see what is behind it. At least part of the title bar always stays on screen, so it can be grabbed back.
- **Resize it** by dragging its bottom-right corner (*Drag to resize*). This is a pointer gesture only.
- Where you put it lasts as long as it is open; closed and opened again, it returns to where its author placed it. Nobody else's screen is affected.
- **Inside a card or dialog, everything stays inside it.** A press that would open another page, choose another asset or window, or look closer at a tile changes what the card shows instead of opening something new over it, and a **Back** arrow (←) appears in its title bar. **Escape** steps back one view at a time, and closes the card when there is nothing to step back to.
- Opening another page from the rail closes every card and dialog: they belong to the page they were opened from.

Worked example. On **Line mimic** the top sealer's **STARVED** lamp is lit. The operator presses the **Checkweigher** box: a card titled **Machine** opens beside it, showing the checkweigher's state and time in state, its rate against its rated speed and the last 30 minutes of its rate, while the mimic behind keeps updating. The card covers the buffer bar the operator wanted, so they drag the card aside by its title bar. **Escape** closes it, and pressing the **Top sealer** box then opens the same page for the top sealer.



The Machine card over the Line mimic page, opened from the Checkweigher box: the checkweigher's state, its rate against its rated speed, its recent rate and its running, blocked, starved and faulted timelines, with the shaded mimic still live behind it

Alarms

The alarm banner and counts

The banner in the header carries the worst unacknowledged alarm on the installation. In the figure under *Live trends* it reads `ALARM Packaging Compressed air low 14s ago`: the demonstration alarm **Compressed air low**, raised when `Packaging/Line/AirPressure` stays below `6.0` bar for 15 s. The **Alarm** count beside it reads `1`.

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

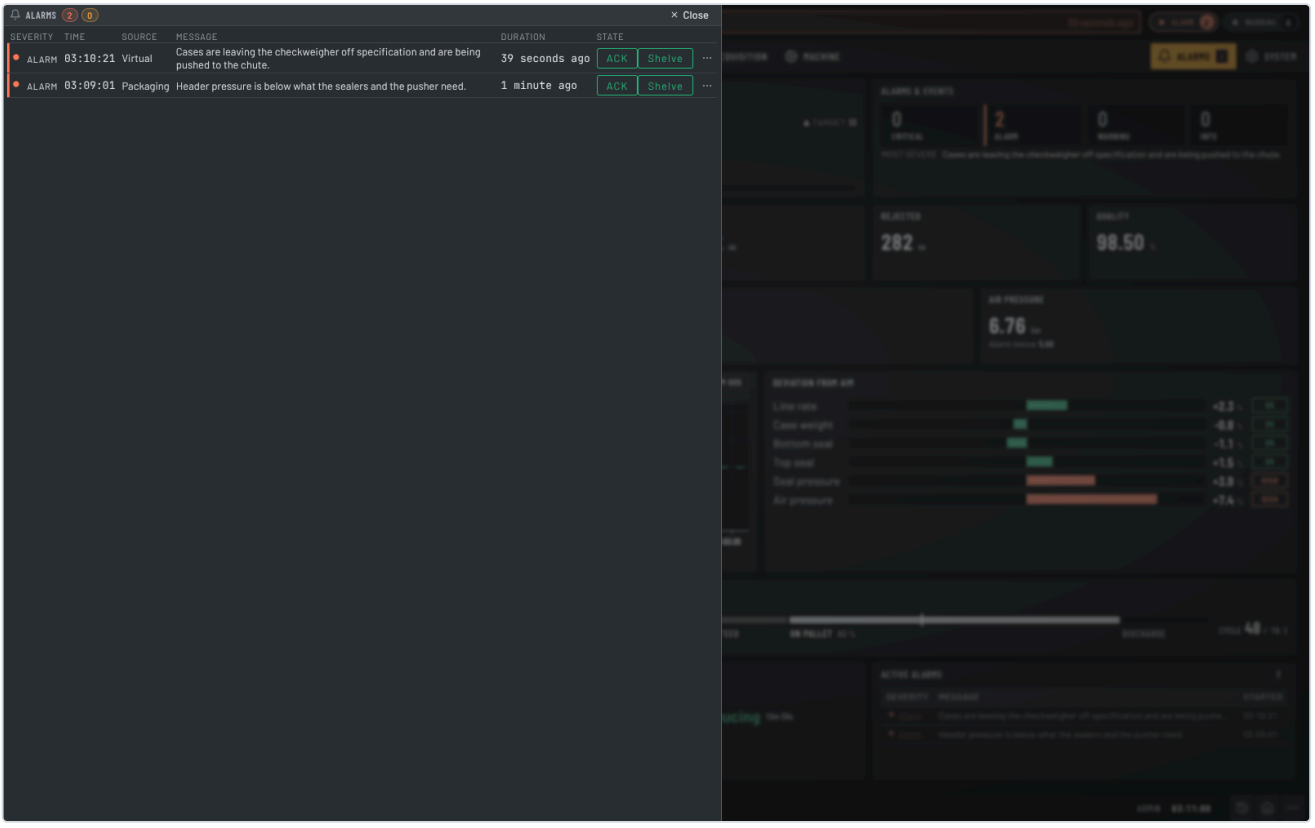
The alarm drawer

Pressing the banner, the counts, or **Alarms** at the right of the rail opens the alarm drawer over whatever page is open, so the operator does not lose their place. The drawer lists the active alarms:

Column	Content
Severity	Critical, Alarm, Warning or Info, as a word and a symbol.
Time	When it was raised.
Source	The asset path, for example `Packaging`.
Message	The alarm's message and, where the alarm rule gives one, the recommended action.

Column	Content
Duration	How long it has been active.
State	**Acknowledged**, or the **ACK** and **Shelve** buttons, or **Active** where the viewer may not act.
⋯	The row menu: acknowledge, shelve, and the details of the signals the alarm names. Also opened by right-click, the Menu key or Shift+F10.

The drawer closes with its **Close** button or **Escape**.

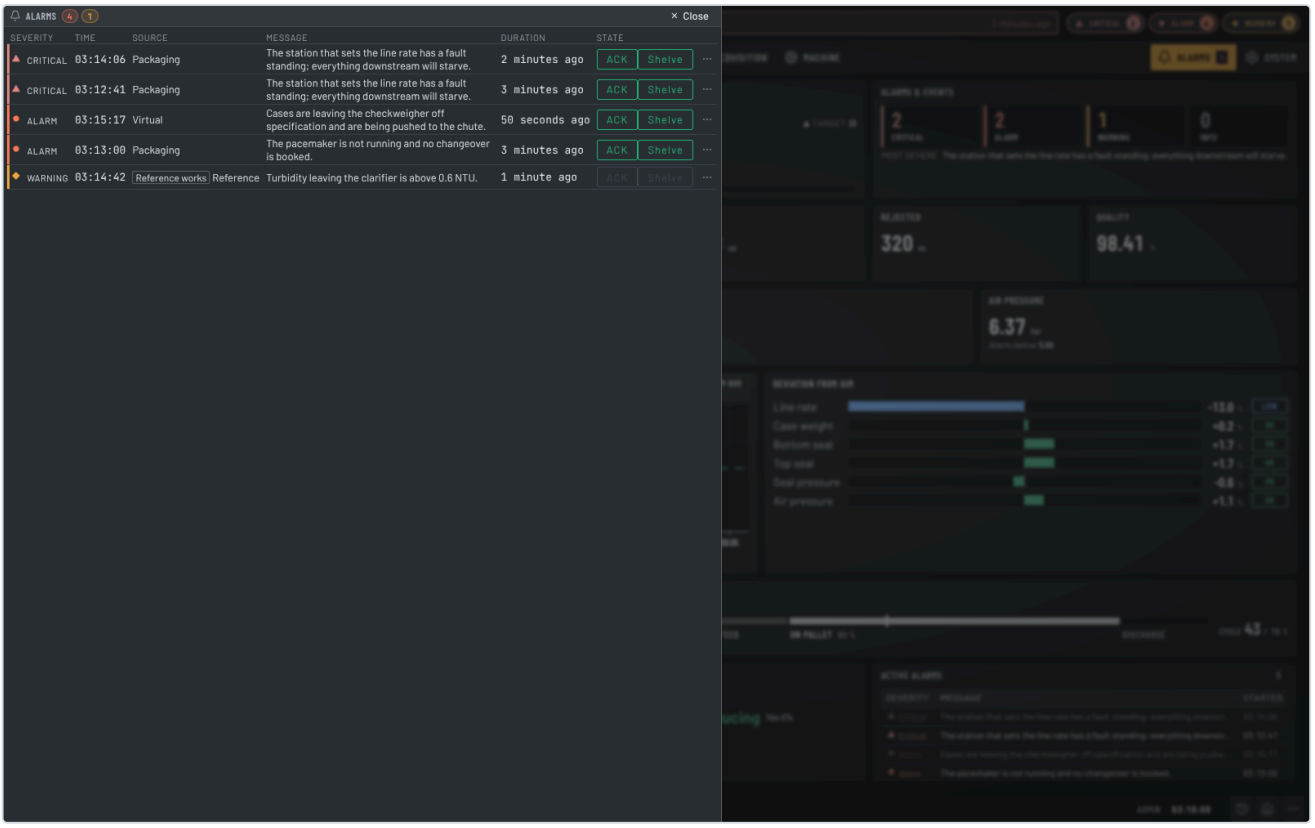


The alarm drawer open over the Overview page: one Warning from Packaging, The label reel is nearly out; the labeler will stop and block the line behind it, nine seconds old, with ACK, Shelve and the row's ⋯ menu

When the drawer shows no **ACK** buttons, a line at its top says why:

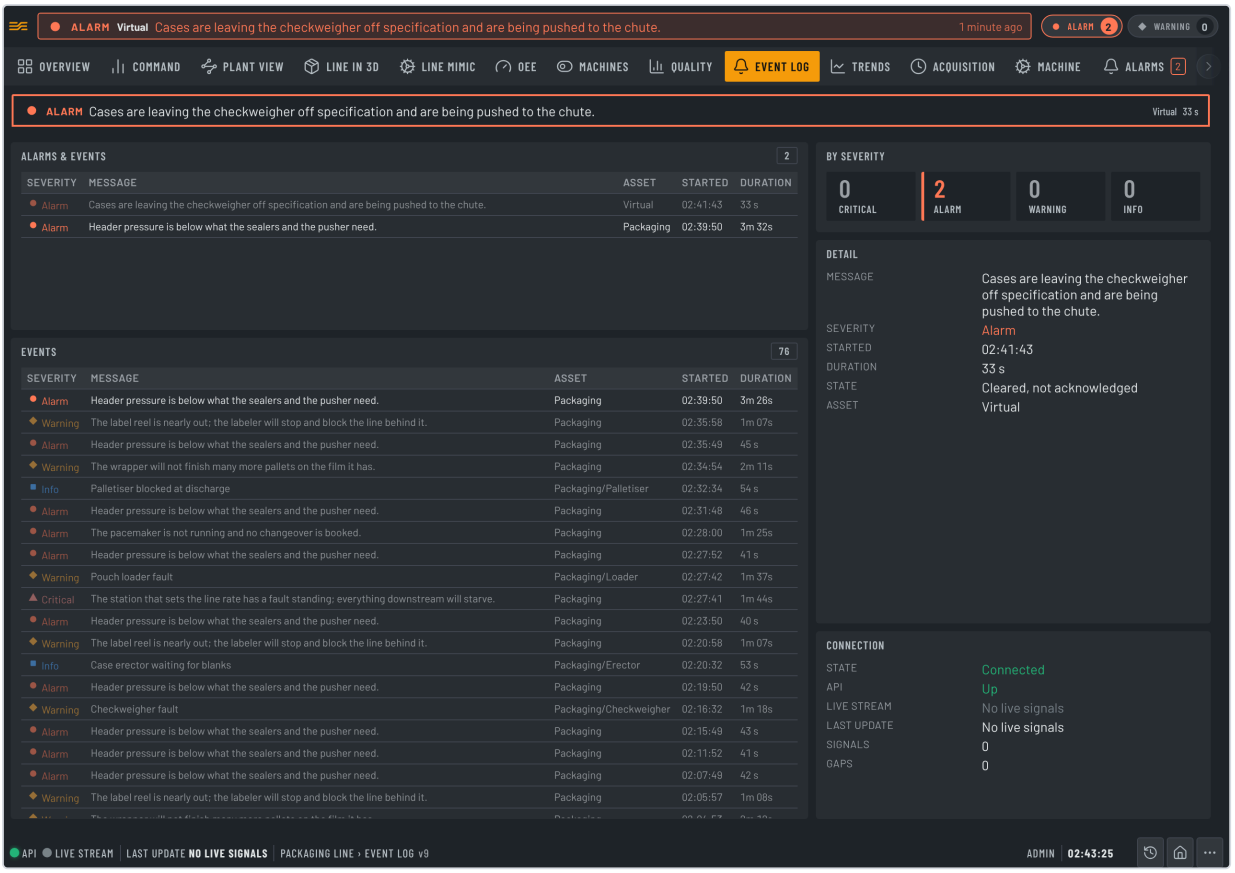
- *Alarms are acknowledged on the node that owns them. This host only reads them.*
- *You are not permitted to acknowledge alarms.*

On a host that reads **linked nodes** (see *Administration ▸ Linked nodes* in the web administration chapter), the drawer also lists the alarms standing on each linked plant. Their **Source** names the plant first, then the asset in that plant, for example **Reference works** `Virtual` for the reference works linked under the prefix `Works`. Their **ACK** and **Shelve** buttons are shown disabled, and hovering them says why: *This alarm belongs to Reference works, which this host only reads. Writes go to the signal's own node: answer it on Reference works's own screens.* When a linked plant does not answer, its alarms are missing from the list, not cleared, and the line across the top of the screen names it — for example *Reference works could not be reached.*.



The alarm drawer on the packaging line with the reference works linked: the packaging line's own header-pressure alarm with ACK and Shelve, and under it the works' supply pressure below its aim, its source Reference works then Virtual, and its ACK and Shelve disabled.

The **Event log** page carries the same information as page components: the banner, a count by severity, the alarm list, the event list with **Severity**, **Message**, **Asset**, **Started** and **Duration**, and a **Detail** panel (*Select an alarm to see its detail*).



The Event log page (captured on an earlier build, where the page was labelled Alarms & events): the banner, the count by severity, the event list with a Line not producing alarm, and the alarm detail panel

Acknowledging

1. Open the alarm drawer.
2. Find the alarm, for example **Compressed air low**.
3. Press **ACK** in its row (or choose it from the row menu).
4. Where the site requires electronic signatures for alarms, enter your password again and confirm what signing means.
5. The row changes to **Acknowledged**, and the banner moves on to the next worst unacknowledged alarm.

Acknowledgement needs the **Acknowledge alarms** permission (`Hmi.AcknowledgeAlarms` , held by the Operator role). It is decided by the node that raised the alarm, as you: the node records your name in its audit trail and every workstation's alarm list shows the alarm answered. It writes nothing to the process.

Shelving and the shelving review

Shelve sets an alarm's rule aside for an hour; the alarm returns by itself when the time runs out. A shelved alarm is still recorded; it simply tells nobody while shelved. Shelving needs the same permission as acknowledging, and is only offered on a host that runs the alarm engine.

Below the active list, when anything is shelved, the drawer shows the **Shelved** review: the alarm, who set it aside (**Set aside by**), the reason where one was given, how long until it **Returns**, and **Bring back** to end the shelving early. An operator coming on shift reads this to tell an alarm that is quiet because the plant is well from one that is quiet because somebody silenced it.

History replay

Replay lets an operator look back at any published page as it was: the same page, the same mimic, driven by the recording instead of the plant.

1. Press the **replay** button (the clock with an arrow, bottom right). Its tooltip reads *Replay the last hour*. The button is absent where the project's settings turn replay off.
2. A band appears above the page: **REPLAY — the recording, not the plant**. Every value on the page is now from the past.
3. Use the replay bar:

Control	What it does
Previous alarm / **Next alarm**	Jumps the cursor to the previous or next alarm in the window.
Play / **Pause**	Plays the window forward from the cursor.
Speed button (`1×`)	Cycles the playback speed: 1×, 2×, 5×, 10×, 60×. At 60× an hour plays in a minute.
Slider	Drags the cursor through the window.
Clock	The instant the page is showing.
Window	15 min, 1 h, 8 h, 24 h, or **Custom…** with a from and to date and **Show**.
Zoom out to the window before this one	Appears after zooming into part of a window.
Downsampled note	Shown when the window was too long to send every sample; it says how coarse the replay is.
Back to live	Ends replay and returns to the plant.

Replay is read-only. Commands and acknowledgements are refused while replaying, with *This screen is replaying the past. Return to live before commanding anything.*, because acknowledging a two-hour-old alarm from a replayed screen acknowledges the wrong thing. A replay left paused and untouched for five minutes returns itself to live, so the next shift is not left reading an old screen.

Worked example. The night shift reports that the line stopped around 02:10. The day operator opens **Line mimic**, presses replay, chooses the **8 h** window, presses **Next alarm** until the cursor reaches **Line not producing**, then plays at 10× from a few minutes earlier and watches which machine's **BLOCKED** lamp lit first. **Back to live** returns the page to the plant.

Commands

Monitoring is the default. A page can carry command controls only where its author added a `WriteTag` interaction (write a value to a PLC tag) or an `AcknowledgeAlarm` interaction. On the demonstration line, **Order target** and **Target rate** on **Command** are `WriteTag` controls.

Pressing a command control asks for confirmation (*Set Order target to 20000?*) where the author required it, and for a value where the control is an operator setpoint rather than a fixed write. The write then has to pass four independent gates:

1. **The host** must have been started with writes allowed and must be acquiring. If not: *Commands are not enabled on this host.* or *This host does not acquire, so it cannot write to the process.*
2. **The operator** must hold the command permission (`Hmi.CommandProcess`). If not: *You are not permitted to command the process.*

3. **The configuration** must allow the write: the connection's **Allow writes**, the tag's **Writable** flag, and the tag's write range (for `Packaging/Line/OrderTarget`, 1 000 to 60 000). A tag the running project does not have gives *The running project has no such connection or tag, so nothing was written.*
4. **The audited write authorizer**: every attempt, allowed or refused, is recorded in the audit trail, the same way a write from the Desktop is.

If the device itself refuses the value: *The device did not accept the value.*

> **The 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.

What an operator may and may not do

The seeded roles decide what the browser offers. Buttons a viewer may not use are not drawn, and the node enforces permissions on every request regardless.

Action	Viewer	Operator	Engineer	Administrator
View published pages, live values and trends	Yes	Yes	Yes	Yes
Replay history on a page (where the project allows replay)	Yes	Yes	Yes	Yes
Acknowledge and shelve alarms	No	Yes	No (unless granted)	Yes
Command the process from a page	No	Yes, behind the four gates	No (granted per person)	Yes, behind the four gates
See automation rules and their runs	No	Yes (read only)	Yes	Yes
Open the HMI Studio, edit and publish pages	No	No	Yes	Yes
Take over another author's page	No	No	No	Yes

An operator may **not**: change a page, change which pages anyone is assigned, change an alarm rule, change an automation rule, or acknowledge while replaying. The Studio is not offered in the ⋯ menu to an account without the permission to use it, and the Automation workspace is not offered in that menu at all; it is reached from the other browser screens, for those who may read it.

Assignments: who sees which pages

When somebody signs in, the web host resolves **one project and a start page** for them from the site's assignments. An assignment has a **Scope**, a **Target**, the **Project** it serves, an optional **Start page** and a **Priority**:

Scope	Target	Example
Everyone	—	The installation default
Role	A role name	`LineSupervisor`

Scope	Target	Example
User	A username	`operator`
Station	A paired station's profile	`line-1-pulpit`
Area	The session's selected area	`Plant/Packaging`

Higher priority wins; ties break toward the more specific scope, then toward the newer assignment. With nothing assigned, every viewer is served the installation's default project.

The demonstration installation's assignments show the pattern:

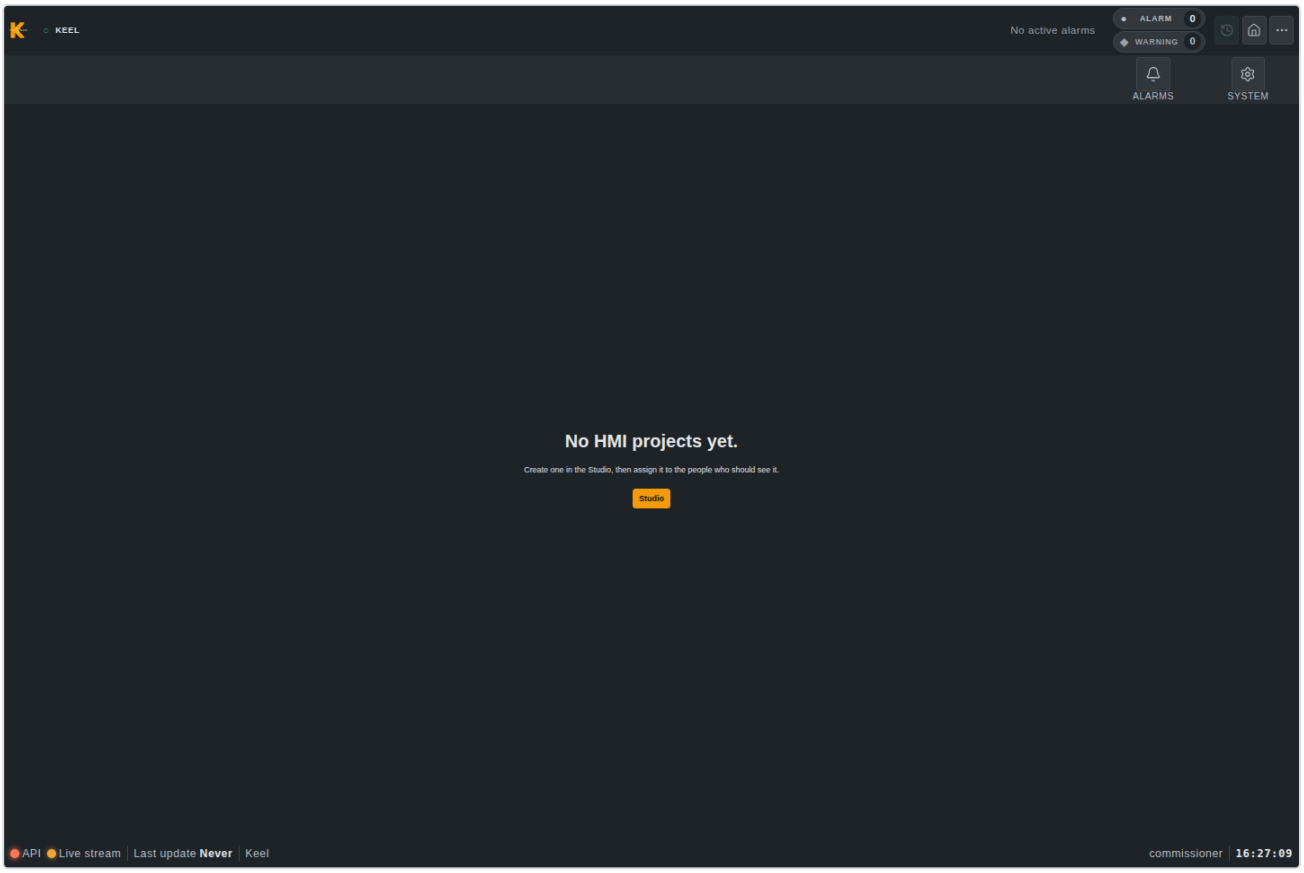
Assignment	Serves	Start page	Priority
Everyone	Packaging line	its default (`overview`)	0
Role `LineSupervisor`	Packaging line	`oee`	10
User `operator`	Packaging line	`command`	30
User `viewer`	Packaging line	`overview`	30
Station `line-1-pulpit`	Packaging line	`line` (Line mimic)	40

So a supervisor lands on **OEE**, the user `operator` lands on **Command**, and whoever signs in at the paired pulpit station lands on **Line mimic**, because the station assignment has the highest priority. Other roles in the same file send the supervisors of the other demonstration plants to their own projects; a person entitled to more than one project gets the **Site** selector.

Assignments are maintained in the HMI Studio; see *The HMI Studio* chapter.

When there is no page to show

Instead of a page, the workspace sometimes carries a heading, a sentence saying why, and, for the person who can fix it, a button. While the HMI is still loading it says **Loading…**.



The operator screen on a freshly commissioned node with nothing authored yet: the header and status bar are drawn, and the workspace reads No HMI projects yet., Create one in the Studio, then assign it to the people who should see it., with a Studio button

Heading	Why	What you can do
This server is not serving a Web HMI.	The Web HMI module is not running on this node. The sentence under it says which of these it is: *The Web HMI module is not installed on this machine. The installer adds it.*; *The Web HMI module on this machine cannot be loaded by this build: its version or its signature does not match.*; *This installation's license does not include the Web HMI.*; *The Web HMI is licensed and this site has not switched it on. It is switched on under Modules.*; *The Web HMI is switched on and is not working. The node's log says why.*; or, for an account that may not use it, *You are not permitted to do that.*	An administrator gets two buttons: **Enable Web HMI** and **Modules**. Anybody else should tell an administrator what the sentence says.
No HMI projects yet.	*Create one in the Studio, then assign it to the people who should see it.* This is where every freshly commissioned node lands.	Somebody who may author pages gets a **Studio** button.
No HMI project is assigned to you.	*A project exists here, but none is assigned to this user, role or station. Assignments are made in the Studio, beside the project they apply to.* When a particular project was asked for, the sentence is the server's reason for refusing that one instead.	Somebody who may author pages gets a **Studio** button; anybody else asks an administrator for an assignment (see *Assignments: who sees which pages*).
No page to show	A project was loaded, but there is no page in it to open.	Choose a page on the rail, if it has any, or ask the project's author.

Enable Web HMI does exactly what **Enable** on **Administration ▸ Modules** does for the Web HMI: it activates a configuration revision, recorded with the comment *Web HMI enabled from the first-run prompt*, which can be rolled back like any other. The button shows that it is busy for a few seconds while the host picks the change up, and the screen then loads the HMI. If the node refuses, its reason appears above the buttons. To turn the module off again, or to see why it cannot be turned on, use **Modules** (see *Modules*).

Somebody with nothing here is sent somewhere useful. When you open the site's bare address (`https://plant-server:8443/`) and there is no HMI for you (one of the first three headings), and you are neither an administrator nor an author but may read history, the browser takes you to **Analytics** instead. An administrator or author is always shown the explanation, because it carries their next step, and so is a paired station. The address `/hmi` always shows the operator screen, explanation and all.

Worked example. After commissioning a new node from `/setup`, the administrator signs in and sees **This server is not serving a Web HMI.** with *The Web HMI is licensed and this site has not switched it on.* They press **Enable Web HMI**; a few seconds later the screen reads **No HMI projects yet.**, as in the figure, and **Studio** takes them to the HMI Studio to import or create the packaging project and assign it.

Stations, kiosks, phones and installed apps

A paired station is a browser bound to a station profile with a pairing code (see *Signing in*). Nobody signs in at it:

- It is served the pages its assignment gives it, through its station, its roles and its area. In the demonstration, the station `line-1-pulpit` lands on **Line mimic**.
- It acts as the station's service account, and it may do only what an administrator ticked for it under **Administration ▸ Stations** (by default, show pages and nothing else). See *Stations*.
- Its screens open in the language set on the station profile.
- The status bar shows the station id where a person's name would be, and **System ▸ Session** has a **Station** line.
- **More ▸ Sign out** ends this screen's pairing; the sign-in card returns, and pairing it again takes a new code from an administrator.

A kiosk is a station set up to show the plant and nothing else around it. The project's author sets this up in the Studio's **Theme and palette** dialog:

Setting	Effect on the screen
Prefer full screen (kiosk)	When the HMI opens it asks the browser for full screen. Browsers grant this only after somebody has pressed something, so in an ordinary browser tab it may not happen until you choose **More ▸ Full screen**. A browser started in its own kiosk mode, or the HMI installed as an app, takes it.
Show header cleared	The header starts closed, behind a **Show header** tab.
Show navigation rail cleared	No rail: the screen stays on its start page (and wherever its components link to).
Show footer cleared	No status bar; **Replay**, **Home** and **More** move to the header.
Operators may switch between light and dark cleared	**More ▸ Theme** is not offered.



The Line mimic as a kiosk, with the packaging project set to Console navy, no header and no footer, the rail as an overlay and the alarm banner as a top band: a Show header tab at the top edge, the menu handle at the top left, and the plant filling the rest of the screen

Where the rail is the overlay drawer (always, on a phone) and the header is closed, the Meelmo mark that opens the drawer is not on screen, so a menu handle is drawn over the top-left corner of the page instead; it opens and closes the drawer in the same way.

On a phone (a window under 768 CSS pixels wide), the rail is always the drawer behind the Meelmo mark, dashboard pages reflow to two columns and scroll, and process canvases open fitted to the screen. Cards and dialogs open at the size their page needs, within the screen.

Installing the HMI as an app. Where the browser offers it, **More ▸ Install** installs the HMI with its own icon and window, without the browser's address bar. **System ▸ Client ▸ Installed as an app** then reads **yes**. The installed app's icon offers three shortcuts: **Alarms** (it opens the page with the id `alarms`, which on the packaging line is **Event log**), **Analytics** and **Studio**.

The keyboard on the operator screens

Every control an operator needs can be reached without a pointer, except the gestures that only make sense with one: dragging a card, turning a 3D model and panning a canvas.

Key	Where	What it does
Tab / **Shift+Tab**	Everywhere	Moves through the header (the alarm banner, the counts, the **Site** selector), the rail's page buttons and **Alarms** and **System**, the page's controls (including each 3D model's part buttons and a canvas's zoom buttons), and the footer's **Replay**, **Home** and ⋯. The ‹ and › buttons at the ends of a top or bottom rail are not in the order; **Tab** scrolls the rail to the button it reaches.
Enter or **Space**	A focused button	Presses it. The alarm banner and the counts open the alarm drawer.

Key	Where	What it does
Escape	Alarm drawer	Closes it.
Escape	Detail overlay, card or dialog	Steps back one view; closes it when there is nothing to step back to. A card or dialog takes the focus when it opens, so **Escape** reaches it straight away.
Escape	⋯ menu	Closes it (or leaves a submenu) and returns the focus to ⋯.
Arrow keys, **Home**, **End**, a letter	⋯ menu	Move through its lines; → opens a submenu (**Language**, **Sign out**), ← leaves it, **Enter** chooses.
Menu key or **Shift+F10**	A row of the alarm drawer	Opens the row's menu: acknowledge, shelve, and **Details**.
Ctrl+K	Everywhere	Opens the command palette (see *The command palette*).
Tab, then arrow keys	The replay bar	Its buttons (**Previous alarm**, **Play**, **Next alarm**, speed, zoom out, **LIVE**) are ordinary buttons; the position slider moves with the arrow keys; the **Window** list is an ordinary list.

The HMI Studio

> **Module: Web HMI.** This chapter covers the HMI Studio, the browser editor for operator screens supplied by the Web HMI module. It is only present on installations licensed for that module.

The HMI Studio is where the operator screens described in *The browser operator HMI* chapter are drawn. It is served by the same web host that serves the operator screens, at `/studio`. You draw a page, bind its components to the node's signals and assets, check it, and publish it.

This chapter covers the Studio's screens and dialogs. What each component on the palette is and what its settings do is in the *HMI component reference*. Uploading images and 3D models, the expression language, page parameters and the full keyboard reference are in *HMI assets, expressions, parameters and keys*.

Two facts explain most of its behaviour:

- **The component catalogue is shared.** The server that validates a page and the browser that draws it read the same catalogue. A page cannot use a component the runtime does not have, and a property the runtime ignores is not offered.
- **Publishing is an activation on the node.** A published page becomes part of the node's activated configuration. That is why publishing moves a version number, is recorded in the audit trail, is served identically to every client of that node, and why rolling back is a real operation rather than restoring a file.

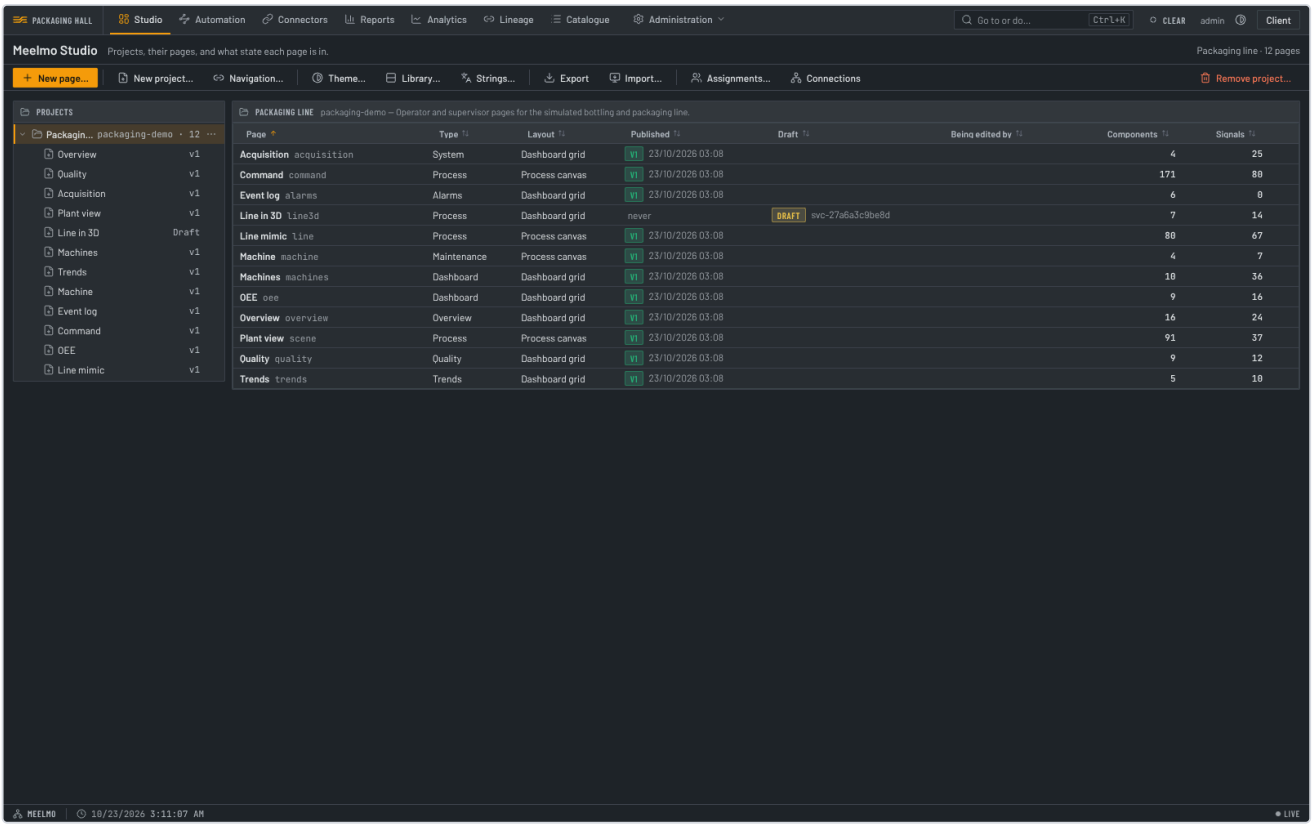
Who may use it

The Studio's permissions are the node's, granted where every other permission is granted (see the *Users, roles and security* chapter).

Permission	Allows
Edit pages (`Hmi.EditPages`)	Opening the Studio and editing drafts
Create pages / Delete pages	Adding and removing pages
Publish pages (`Hmi.PublishPages`)	Publishing a draft
Roll back pages	Activating an earlier published version
Manage templates / composites / themes	**Save as template**, **Save as reusable component**, **Theme**
Assign pages	Maintaining **Assignments**
Break edit lease	**Take over editing** of a page somebody else holds

The seeded **Engineer** role holds all of these except *Break edit lease*; the **Administrator** holds all of them; **Operator** and **Viewer** hold none, so the Studio does not open for them. Editing and publishing are separate on purpose: an author can work all day on drafts without changing what the plant is looking at.

The project browser



The Studio project browser: the Packaging line project and its twelve pages, with each page's type, layout, published version, draft state, editor, component count and signal count

Open **Studio** in the top navigation. The first screen, `/studio`, lists the projects on the left and, for the selected project, its pages in a table.

The toolbar along the top acts on the project:

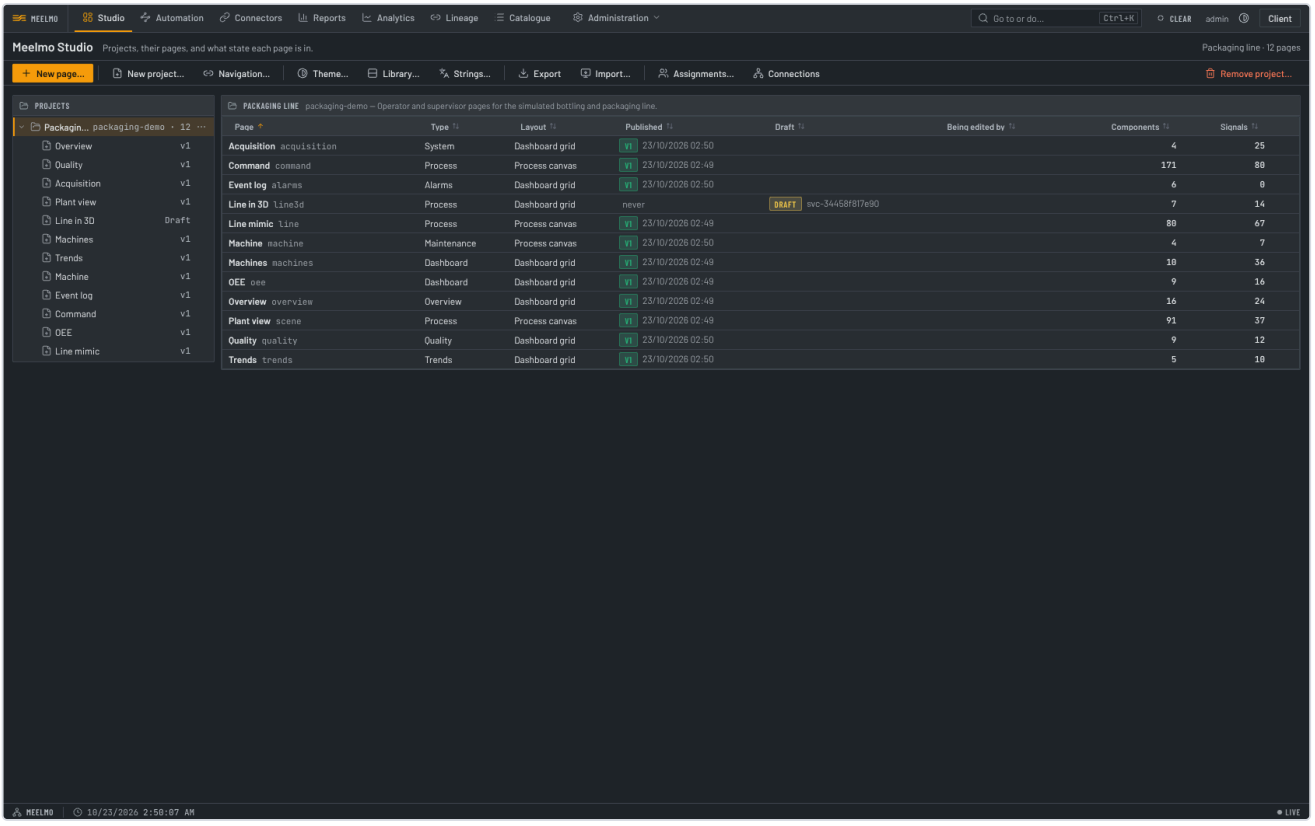
Button	What it does
New page…	Adds a page to the selected project (see *Creating a page*).
New project…	Creates a project. Asks for a **Project name** and a **Project id**.
Navigation…	Edits the page rail operators see.
Arrange missing resolutions	Gives every page of the selected project an arrangement at every resolution it lacks one for (see *Layout and resolutions*). Nothing already arranged moves; each page it completes is saved as a draft, to be published as any other edit. A page somebody has open is left alone and named.
Theme…	Edits the project's colours, type and shell layout.
Library…	Lists templates and reusable components.
Strings…	Edits the project's string table (translations).
Export	Downloads the project as a `.meelmohmi` bundle.
Import…	Loads a `.meelmohmi` bundle, as a new project or replacing one (*Replace the project already here*); the result says which pages were **Published** and which were **Left as drafts**, and lists **Missing media**.
Assignments…	Decides which viewers get which project and start page.
Connections	Takes an administrator to **Admin ▸ I/O**, where the node's connections are configured and tested.

Button	What it does
Remove project…	Removes the project, every page in it and every assignment that serves it. It asks first and cannot be undone.

The page table:

Column	Meaning
Page	Name and id (`Overview` `overview`).
Type	The page's purpose: Overview, Dashboard, Process, Alarms, Trends, Quality, Maintenance, System, and so on.
Layout	**Dashboard grid** or **Process canvas**.
Published	The live version (`V1`) and when it was published, or *never*.
Draft	**DRAFT** and the draft's author when unpublished changes exist. In the figure, **Line in 3D** has a draft and has never been published, so it is not on the operators' rail.
Being edited by	Who holds the page's edit lease now.
Components, **Signals**	How many components the page carries and how many distinct signals it reads. The **Command** page has 171 components and 80 signals.

The project tree on the left lists the same pages with their published version, and opens a page in the editor. With no project at all the Studio says *No HMI projects yet. Create one to begin*.



The project browser on an earlier build, with Edit and Open beside each page

On earlier builds each row carried **Edit** (open in the editor) and **Open** (view it as an operator would) buttons, as above. They are now on each row's menu.

Menus on projects and pages

Every project in the tree and every page in the tree or the table has a menu. Open it with a right click, with the `Menu` key (or `Shift` + `F10`), or from the row's ⋯. The menu holds only what the toolbar already does, applied to that row, so you do not have to select the project first.

The menu on a **project** names the project at the top, then:

Entry	What it does
New page…	Selects the project and opens *New page* for it.
Navigation…	Opens the navigation editor for that project.
Theme…	Opens the theme editor for that project.
Library…	Opens the library for that project.
Strings…	Opens the string table for that project.
Export	Downloads that project as a `.meelmohmi` bundle.
Remove project…	Removes the project after asking. Shown only with the delete pages permission.

Without the create pages permission, the project menu shows only the project's name.

The menu on a **page** names the page, then:

Entry	What it does
Edit	Opens the page in the editor, at `/studio/{project}/{page}`.
Open	Opens the published page as an operator receives it, at `/page/{page}`. It is unavailable for a page that has never been published, and says *It has never been published.*
Delete page…	Deletes the page after asking. Shown only with the delete pages permission.

Deleting pages. Select one or more pages in the table (Ctrl-click, Shift-click) and press **Delete page** (or **Delete N pages**) in the page group of the toolbar, or use a page's menu. The question names the pages and what names them: the project's **start page** (the project will then have none), a **navigation entry** that opens one (it is taken out), and an **assignment** that lands somebody on one (it starts on the project's start page instead). The page's draft and every published version go with it, operators stop seeing it, and it cannot be undone from the Studio. A page somebody else is editing is refused, naming them; the others go.

Creating a page

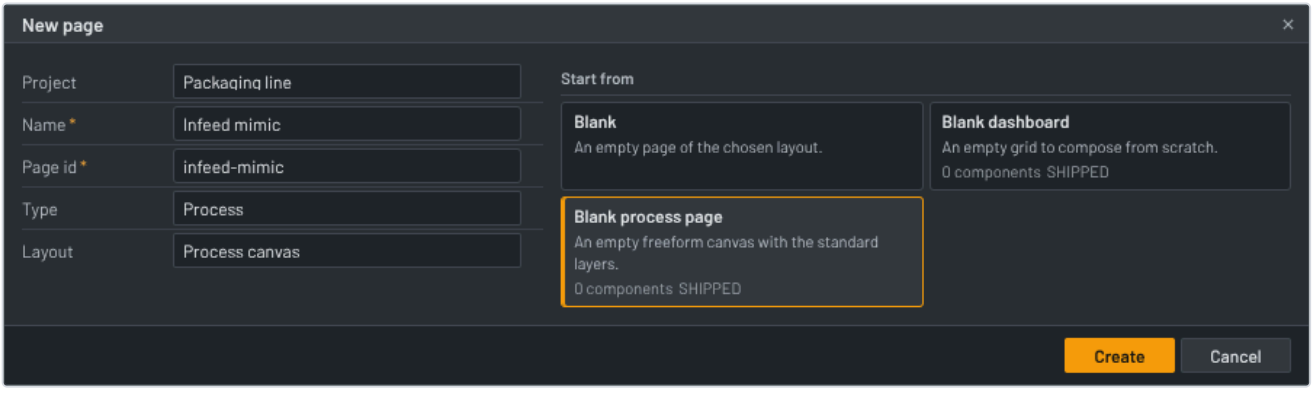
1. Select the project and press **New page…**.
2. Enter a **Name** (for example `Line overview`) and a **Page id**.
3. Choose a **Type**.
4. Choose the layout:
   - **Dashboard grid**: a 12-column grid where components occupy cells. The right default for dashboards, KPI screens and lists.
   - **Process canvas**: a freeform drawing surface with world coordinates, for mimics that must look like the plant.
5. Under **Start from**, choose **Blank** (*An empty page of the chosen layout*), a shipped template, or a template your site saved.
6. Press **Create**. The page opens in the editor as a draft.

A new grid page — blank, from a template, or a copy of another page — is arranged at every resolution at once: the Studio derives the Laptop, Tablet, Phone and Large display layouts from the Full HD one and marks each **Auto** (see *Layout and resolutions*). A copy keeps any resolution the page it came from had arranged by hand.

The layout kind cannot be changed later: turning a grid into a canvas would move every component, which is a new page.

The **New page** dialog:

Field	Meaning
Project	The project the page goes into. It starts on the project selected in the browser.
Name	Required. What the page is called on the rail and in the browser.
Page id	Required. The id in the page's address (`/page/{id}`). It follows the name as you type (`Line overview` gives `line-overview`) until you edit it yourself. **Create** stays disabled until the id is well formed.
Type	One of the thirteen page types below. Choosing a type sets **Layout** to that type's default and clears the template you had picked.
Layout	**Dashboard grid** or **Process canvas**. You may change it after choosing the type.
Start from	**Blank**, or a template card. Each card shows the template's name, description, component count and *shipped* for a built-in one. Only templates of the chosen type are listed, plus templates in the category `general` .



The New page dialog for the Packaging line project: Name Infeed mimic, Page id infeed-mimic, Type Process, Layout Process canvas, and under Start from the Blank, Blank dashboard and Blank process page cards, with Blank process page chosen

Page types

A page's type says what the page is for. It does not change how the runtime draws the page. It decides three things in the Studio: the layout **New page** proposes, which palette categories open first when you edit the page, and which templates **Start from** offers. You can change the type later in **Page settings**; the layout stays as it was.

Type	Default layout	Palette categories opened first	Shipped templates of this type
Overview	Dashboard grid	Execution and context, Values and KPIs, Alarms and events, Navigation	Operator overview
Dashboard	Dashboard grid	Values and KPIs, Gauges and indicators, Trends and analytics, Execution and context	Blank dashboard, Supervisor overview, Equipment detail

Type	Default layout	Palette categories opened first	Shipped templates of this type
Process	Process canvas	Shapes and symbols, Process, Gauges and indicators, Navigation	Blank process page
Alarms	Dashboard grid	Alarms and events, Navigation	Alarm page
Events	Dashboard grid	Alarms and events, Execution and context	—
Trends	Dashboard grid	Trends and analytics, Values and KPIs	Trend page
History	Dashboard grid	Trends and analytics, Values and KPIs	—
Production	Dashboard grid	Values and KPIs, Trends and analytics, Execution and context	Production summary
Quality	Dashboard grid	Values and KPIs, Trends and analytics, Execution and context	Quality summary
Energy	Dashboard grid	Values and KPIs, Trends and analytics, Gauges and indicators	Energy summary
Maintenance	Dashboard grid	Values and KPIs, Alarms and events, Execution and context	Maintenance summary
System	Dashboard grid	Values and KPIs, Information	—
Custom	Dashboard grid	Containers, Values and KPIs	—

Blank dashboard and **Blank process page** are filed under `general`, so they are offered for every type. The other categories of the palette are still there, closed, below the ones the type opens.

A process canvas starts with eight named layers, back to front: **Background**, **Equipment**, **Process connections**, **Measurements**, **States**, **Alarms**, **Navigation hotspots** and **Labels**. A dashboard grid has no layers.

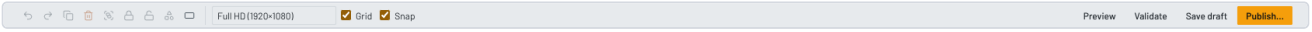
The page editor



The page editor on the Overview page: the toolbar, the Components palette on the left, the canvas with live values in the middle, Page settings on the right, and the validation strip along the bottom

Press a page in the tree (or **Edit**) to open `/studio/{project}/{page}`. The editor has five areas: a toolbar across the top, a three-tab panel on the left, the canvas in the middle, the inspector on the right, and a validation strip along the bottom.

The toolbar



The page editor toolbar: undo, redo, duplicate, delete, group, lock, save as reusable component, the breakpoint selector, Grid and Snap, and Preview, Validate, Save draft and Publish

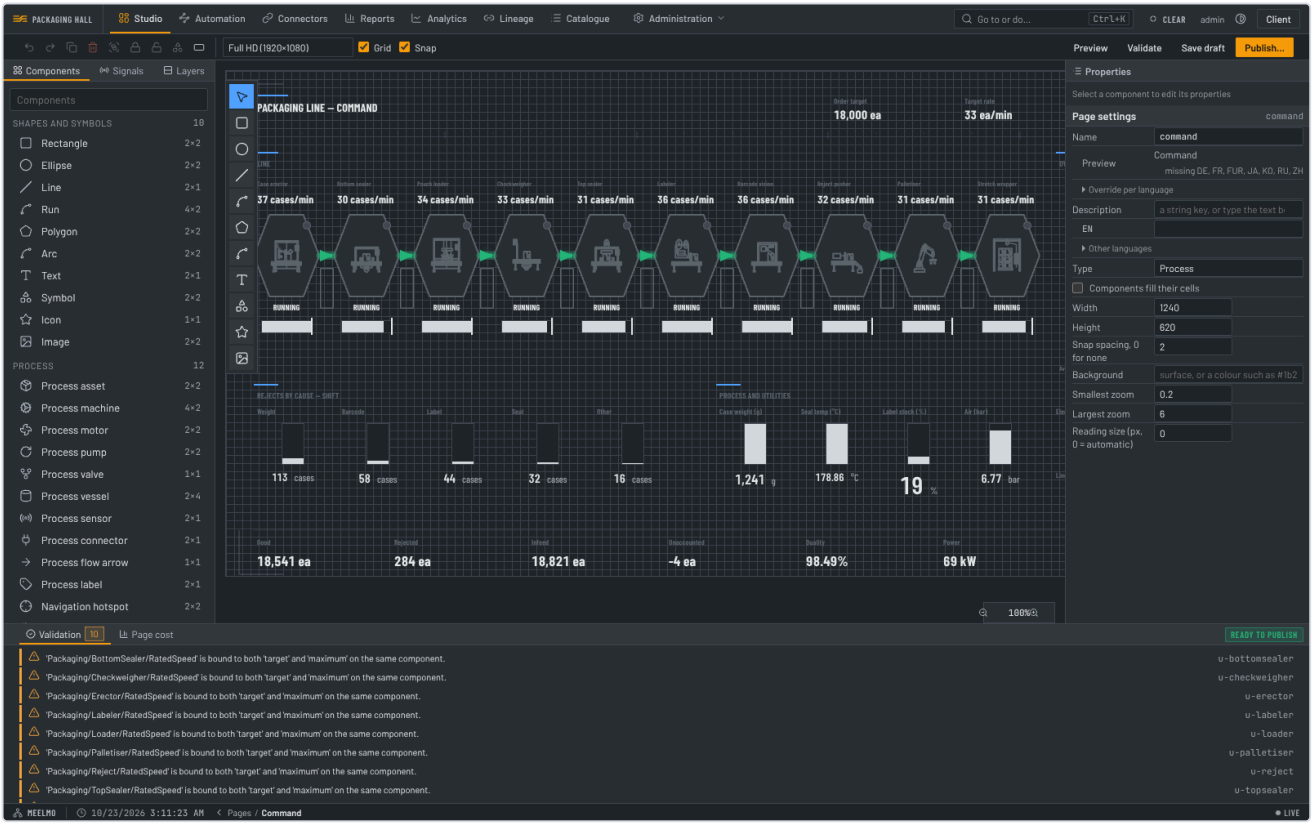
Left to right:

Control	What it does
Undo / **Redo**	Each tooltip names the action it will reverse or repeat.
Duplicate	Copies the selection in place.
Delete	Removes the selection.
Group	Groups two or more components so they move and scale together.
Lock / **Unlock**	Fixes or frees the position of everything selected.
Save as reusable component	Turns the selection into a composite (see *Composites and templates*).
Save as template	Shown to somebody who may manage templates; saves the whole page as a starting point for new pages.

Control	What it does
Breakpoint selector	`Full HD (1920×1080)` and the other breakpoints, each with its nominal size.
Arrangement badge	Beside the selector, on any breakpoint other than Full HD: **Auto** (arranged by the Studio from Full HD), **Edited** (arranged by hand, kept when the others are re-arranged) or **Incomplete** (a component has no place here yet; saving fills it). On a process page, **Too small** where the drawing would be scaled below a readable size.
⇲ **Derive from Full HD**	On any breakpoint other than the design one, arranges that breakpoint afresh from the Full HD layout and marks it **Auto** again.
Arrange missing resolutions	Gives every resolution with no arrangement, and every component a resolution has no place for, an arrangement. Nothing already arranged moves. Disabled when nothing is missing.
Grid, **Snap**	Show the grid; snap to it. Independent of each other.
Preview	Shows the page as an operator receives it (see *Preview and simulate*).
Ask Assistant	Opens the **Assistant** tab beside **Properties** in the right-hand panel. Shown only where the Studio Assistant is licensed and switched on; see the *Studio Assistant* chapter.
Validate	Runs validation now.
Save draft	Stores your work without changing what any operator sees.
Publish…	Validates and publishes. Disabled without the publish permission.

A **Draft** badge beside the page name means there are unsaved changes.

The canvas



The page editor on the Command process page: the drawing tools down the left edge of the canvas, the Shapes and symbols and Process palette categories, the canvas page settings, and ten validation warnings

The canvas shows **real values**. It reads them once when the page opens rather than subscribing, so it is representative without being a second live client. Composing against real readings is the point: a tile sized for `42` is the wrong size for `1 284.6`.

On a grid page, drag components into cells and resize them by their handles. On a process canvas, a vertical toolbar at the left edge of the canvas holds the drawing tools, and a zoom control sits at the bottom right.

Tool	Key	Draws
Select	`V`	—
Rectangle	`R`	Rectangle
Ellipse	`E`	Ellipse or circle
Line	`L`	Straight run
Run	`P`	Run with corners
Polygon	`N`	Closed outline
Arc	`K`	Sector or ring segment
Text	`T`	Label
Symbol	`S`	One of the shipped equipment symbols (tank, silo, hopper, pump, motor, fan, valve, conveyor-type equipment and more)
Icon	`I`	An icon from the shared library
Image	`M`	A project image

On the canvas: `Ctrl` +arrow rotates the selection by 1°, `Ctrl` + `Shift` +arrow by 15°; `Ctrl` +wheel zooms about the pointer; a middle-drag pans; `Esc` puts the tool down. The letter keys pick a tool only on a process canvas. Every other key the editor takes is listed in *Keys in the Studio*.

Drawing a shape is one gesture. Pick the tool, then drag out the rectangle the shape should fill. A click without a drag still draws the shape, at the type's own size. As soon as the shape is drawn the tool goes back to **Select**, so drawing six valves is six picks of the tool. New shapes go onto the **Equipment** layer, in front of everything already on that layer.

- **Symbol** and **Icon** show a list beside the tool strip while they are picked. Choose the symbol (for example *Tank*, *Pump*, *Checkweigher* or *Powered roller conveyor*) or the icon before you drag. The symbols are listed in *The symbol library*.
- **Line**, **Run** and **Polygon** start as a default outline inside the rectangle you dragged. Select the shape to see its vertices: drag a vertex to move it, double-click a vertex to remove it, and click the **+** at the middle of a segment (*Add a vertex here*) to bend the run there.
- A selected shape has a rotation handle (*Drag to rotate; hold Shift for 15° steps*). Double-click the handle to set the rotation back to 0°.

Dropping a signal on a shape is the fastest way to animate it. The Studio adds the animation that kind of value most likely wants, and selects the shape so you can adjust it:

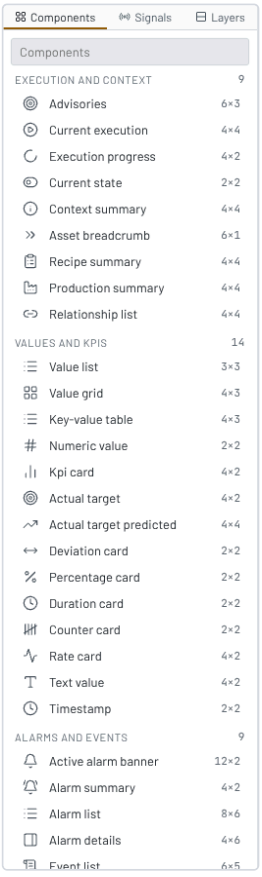
You drop	On	The Studio adds
Any signal	A **Text** shape	**Text**: the shape shows the value.

You drop	On	The Studio adds
A boolean	A closed shape (rectangle, ellipse, polygon, arc, symbol) or an icon	**Fill colour** with two bands: *On* (not 0, Running colour) and *Off* (0, Unavailable colour).
A boolean	A line or run	**Outline colour** with the same two bands.
A number	A closed shape	**Fill level**: 0 to 100 fills the shape from empty to full.
A number	A line, run or icon	**Outline colour** with the *On* and *Off* bands, to retune.

Nothing is added for an image, or for a text-valued signal on anything but a **Text** shape; the shape is selected so that you can add an animation by hand. Dropped anywhere else, the signal binds to the first empty slot of the selected component; with nothing selected, it adds a **Numeric value** bound to it.

The left panel: Components, Signals and Layers

Components is the catalogue, by category, each type with its default size in grid cells (`2×2` , `4×2`). Type in the **Components** box to filter. Drag a type onto the canvas, or click it to add it.



The component palette, by category

Category	Examples of what it holds
Containers	Card, panel, section, tabs, accordion, divider, spacer
Values and KPIs	Numeric value, KPI card, actual target, deviation card, percentage card, counter card, rate card, text value, timestamp, value list, value grid

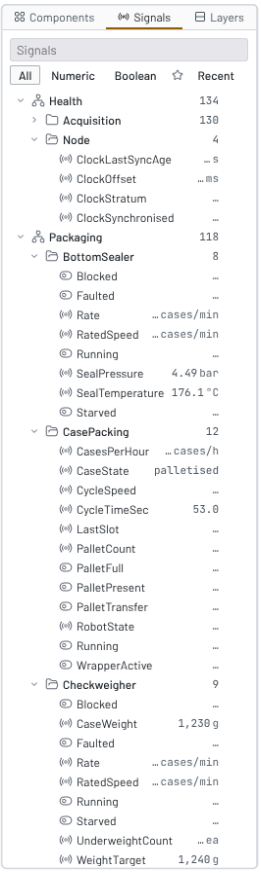
Category	Examples of what it holds
Gauges and indicators	Linear gauge, vertical gauge, radial progress, range gauge, segmented gauge, state indicator, traffic light, quality indicator, progress indicator
Trends and analytics	Live trend, historical trend, multi-signal trend, mini trend, live scope, data logger, multiplot, stacked trend, bar chart, Pareto chart, distribution chart, boolean track, state timeline, execution timeline, statistics card, run overlay, line time account
Alarms and events	Active alarm banner, alarm summary, alarm list, alarm details, event list, event timeline, severity counter, severity filter, asset filter
Process	Process asset, machine, motor, pump, valve, vessel, sensor, connector, flow arrow, label, navigation hotspot, route diagram, 3D model
Shapes and symbols	Rectangle, ellipse, line, run, polygon, arc, text, symbol, icon, image (process canvas only)
Execution and context	Advisories, current execution, execution progress, current state, context summary, asset breadcrumb, recipe summary, production summary, relationship list
Navigation	Area selector, asset selector, asset link, page link, time range selector, trend legend
Information	Connection health, help button, icon, image, legend, text

Signals is the node's signal catalogue — exactly the signals the node's activated revision has: the tags it acquires, the signals it computes and its own health — grouped by path (`PACKAGING/CHECKWEIGHER`), with each signal's current value and unit beside its name. Filter with **All**, **Numeric**, **Boolean**, ★ (favourites) or **Recent**, or type in the **Signals** box. The current value is the quickest way to tell a signal that is producing from one that is merely defined.

The list follows the node. When somebody activates a revision that removes a connection or adds tags, an open Studio shows the change within about ten seconds, without being reloaded. A signal the activated revision no longer has — a removed connection's tags, a renamed tag's old name — is not offered, even though the historian still holds its history.

Long names. The panel is made for tags browsed from an OPC UA server, whose names and node ids are long (`ns=2;s=Gateway|…|ZONE_PRS_SP_PREV`):

- **The search** finds every word you type, ignoring case, anywhere in a signal's path, its display name, its source address or its description, so `zone prs` finds a tag from the middle of its path. On a large node it is applied a moment after you stop typing. Above the tree the panel says how many signals it shows of how many there are (*504 of 3,042*).
- **Group by path** (the folder button beside the search, on by default) groups a tag under its connection and group by its own path — the parts of a name written `Area.Line.Speed` or `Area|Line|Speed` — or, for a plain name such as `ZONE_DB_Pos` , by the folders of its source address, so seven thousand browsed tags are the server's own folders, each collapsible, rather than one list. A folder holding a single folder is drawn as one (`Gateway|Plant`). Off, the tree is connection, group and tag, as before. The choice is kept in this browser.
- **A name too long for its row is shortened in the middle**, never at the end: the start and the name stay (`Gateway|…|ZONE_PRS_SP_PREV`), and the whole of it is the row's tooltip and what a screen reader reads.
- **Each tag has an expander** — the chevron at the end of its row, or `→` on the keyboard (`←` closes it) — that shows, wrapped under the row, its whole path, its display name, its source address (the OPC UA node id), its type and unit, and its description.
- **The panel can be widened.** Drag its right edge, or focus the edge and press `→` and `←` (`Shift` for larger steps, `Home` and `End` for the limits). The width is kept in this browser, for you, the next time you open the Studio.
- The tree draws only the rows on screen, so it stays quick with ten thousand signals.



The signal browser, grouped by machine, with each signal's current value and unit

With a component selected, the bottom of the **Signals** tab lists that component's binding slots under its name (for example *Linear gauge — Signals*), each with what it is bound to now and **REQUIRED** where the slot must be filled. Press a slot to make it the target. Then `Enter` or a double click on a signal binds the signal to that slot; with no slot chosen, it binds to the first slot that is still empty. A signal you bind or add goes to the top of **Recent**, which keeps the last twenty.

The signal menu. Every signal in the tree has a menu: a right click, the `Menu` key or `Shift` + `F10`, or the row's ⋯. It names the signal at the top, then:

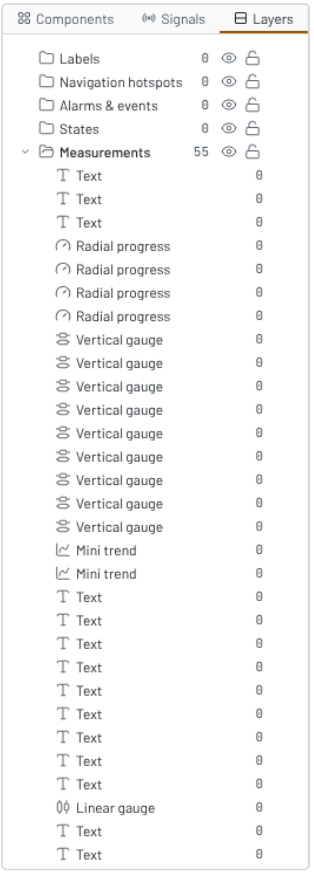
Entry	What it does
Bind to the selected component (`Enter`)	Binds the signal as `Enter` does. Unavailable, with *Select a component on the canvas first.*, when nothing is selected, and on a read-only page.
Add as a value tile	Adds a **Numeric value** to the page, bound to the signal through its **Actual value** slot, whatever is selected.
Favourite	Ticks or unticks the signal as a favourite. Favourites are what ★ shows, and they are kept in this browser for the next time you open the Studio.
Copy the path	Copies the signal id (for example `Packaging/Checkweigher/CaseWeight`) to the clipboard, and says *Copied {path}*. Where the browser refuses, it says *This browser would not copy it. Select the path and copy it by hand.*

Layers lists a process page's named layers in draw order, front at the top. Under each layer are the components on it, front first, each with its z-order on the right; a component shows its **Title**, or its type where it has none. On a grid page the tab says *Layers apply to process pages.*

Each layer row has two toggles for the pointer:

- The **eye** (*Hide while editing* / *Show*) hides the layer on the canvas, and the layer's name dims. The canvas leaves the layer's components out, so you can work on what is underneath; they cannot be clicked, swept or selected with `Ctrl` + `A` while hidden. The mark is saved with the page like any other edit, but it only affects editing: **Preview** and the operator screens draw every layer, hidden or not.
- The **padlock** (**Lock** / **Unlock**) locks the layer, and the mark is saved with the page. The canvas passes a press on a locked layer's component through to whatever is above or around it, so it is not selected by a click, a sweep or `Ctrl` + `A`, and nothing on the layer can be moved, resized, turned, aligned, nudged with the arrow keys or deleted until the layer is unlocked. You can still select a component on a locked layer in this tree and change its settings in the inspector. Use it to keep a background mimic still while you work over it. Like the eye, it only affects editing.

Click a component in the tree to select it on the canvas; `Ctrl`+click adds or removes one row, and `Shift`+click selects a range. A layer row does not select what is on it when you click it, so that opening its menu does not select fifty components. To select everything on a layer, press `Enter` or double-click the layer row, or use its menu.



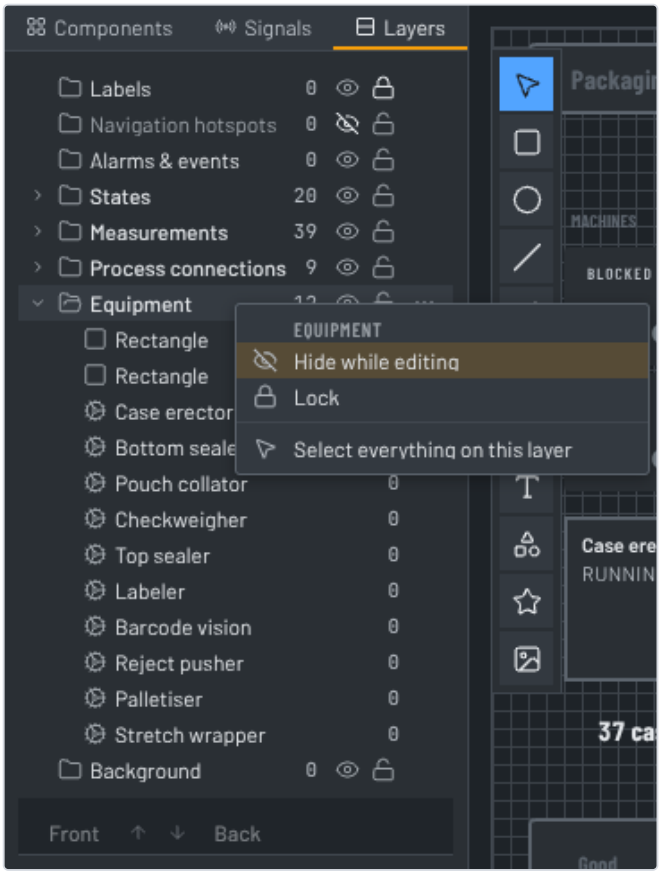
The Layers panel on a process page: Labels, Navigation hotspots, Alarms & events, States and Measurements, with the components on Measurements listed

Under the tree are the controls that act on the selection:

Control	Shown	What it does
Front	Always; needs a selection	Brings the selection to the front of its layer.
Bring forward (up arrow)	Always; needs a selection	Moves the selection one step towards the front.
Send backward (down arrow)	Always; needs a selection	Moves the selection one step towards the back.

Control	Shown	What it does
Back	Always; needs a selection	Sends the selection to the back of its layer.
Align the selection	Two or more selected	Six buttons: **Align left edges**, **Align centres across**, **Align right edges**, **Align top edges**, **Align centres down**, **Align bottom edges**.
Distribute across	Three or more selected	Spaces the selection evenly from left to right: the leftmost and rightmost stay put, and the centres of the others are set at equal steps between them.
Distribute down	Three or more selected	The same, from top to bottom.
Layer	One component selected	Moves that component to another layer.

The menu on a **layer** row names the layer, then **Hide while editing** (or **Show**), **Lock** (or **Unlock**), and **Select everything on this layer** (unavailable on an empty layer).



The Layers tab on the Line mimic page with States, Measurements and Process connections folded: Navigation hotspots hidden (its eye struck through and its name dimmed), Labels locked, and the menu of the Equipment layer open with Hide while editing, Lock and Select everything on this layer

The menu on a **component** row acts on the selection; if the row is not selected, it becomes the selection first. It offers **Bring to front**, **Bring forward**, **Send backward**, **Send to back**, and **Move to layer**, a submenu of the page's layers with the current one ticked. **Move to layer** is unavailable when the page has only one layer. On a read-only page every entry is unavailable.

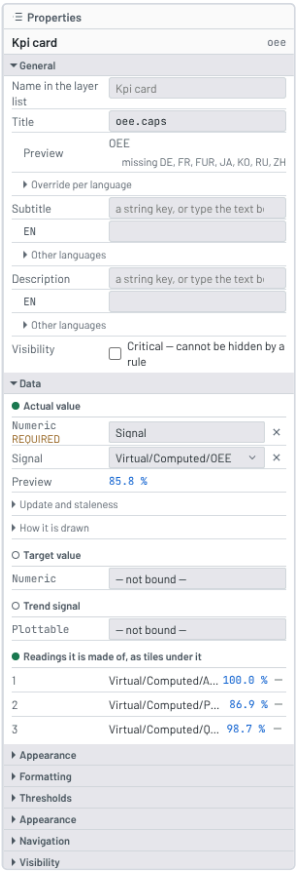
Grouping and locking

These apply to both kinds of page:

- **Group** on the toolbar (`Ctrl` + `G`) groups two or more selected components, so they move and scale together. Clicking one member selects the whole group; hold `Alt` while clicking to select one member alone and edit it in place. `Ctrl` + `Shift` + `G` ungroups every group the selection touches.
- **Lock** and **Unlock** on the toolbar fix or free the position of everything selected. A locked component shows no resize, rotation or vertex handles. **Lock position** under **Advanced** does the same for the one component shown in the inspector.
- **Duplicate** (`Ctrl` + `D`) copies the selection in place; **Delete** (`Delete`) removes it.

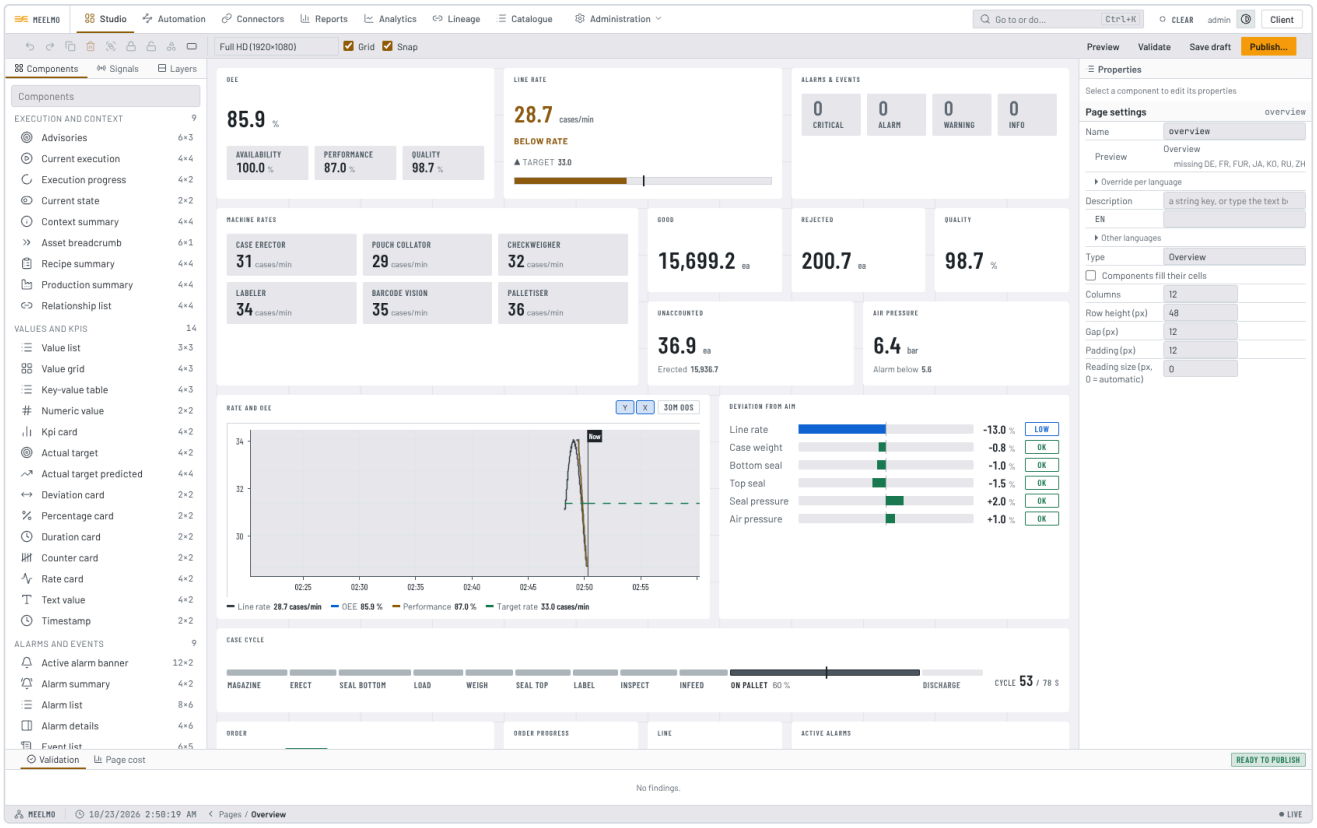
The inspector

The right-hand panel edits whatever is selected, and names the type and id at the top (for example `kpi-card` / `oee`). It is generated from the component's own description, so a component only offers what it can actually do, and every field has the right kind of editor: number boxes with the property's own minimum, maximum and step, lists of choices, colour pickers limited to the theme's semantic colours, duration boxes that accept `15m` , `1h` , `24h` . There is no JSON editing; the serialized form under **Advanced** is read-only.



The inspector on the OEE KPI card: General with the Title key oee.caps, and Data with the Actual value slot bound to Virtual/Computed/OEE

With nothing selected, the inspector says *Select a component to edit its properties* and shows **Page settings**: **Name**, **Description**, **Type**, **Components fill their cells**, **Reading size (px, 0 = automatic)** (one size for every reading on the page: automatic is the largest at which all of them fit), and for a grid page **Columns**, **Row height (px)**, **Gap (px)**, **Padding (px)** (the Overview page uses 12, 48, 12, 12). For a process page it shows **Width**, **Height**, **Snap spacing, 0 for none**, **Background**, **Background picture**, **Smallest zoom** and **Largest zoom** (the Command page is 1240 × 620 with zoom from 0.2 to 6).



The page editor in the light theme on the Overview page with nothing selected: the Components palette on the left, Page settings with Name, Type, Columns, Row height, Gap and Padding on the right, and READY TO PUBLISH in the validation strip

The sections of a selected component:

Section	Holds
General	**Name in the layer list**, **Title**, **Subtitle**, **Description**, and **Critical — cannot be hidden by a rule**.
Data	One row per binding slot (see *Binding signals*). Shapes show **Driven by** instead.
The type's own groups	The settings that belong to this type, for example **Scale minimum**, **Scale maximum**, **Show the scale** on a linear gauge.
Formatting	**Unit override**, **Decimal places**, **Group thousands**, **Show + on positives**, **Compact large numbers**, **Text when there is no value**, **Multiply by**, **Show as**.
Thresholds	Value bands (see below).
Appearance	**Show header**, **Show border**, **Show quality**, **Show timestamp**, **Background**, **Padding**, **Value size**, **Alignment**, **Size to cell** (*Page default*, *Fixed* or *Fill*: whether the component's reading and text scale to its cell).
Navigation	What pressing the component does.
Visibility	The rule deciding whether it is drawn.
Responsive	Placement per breakpoint on a grid page; position, size, rotation and zoom band on a canvas.
Advanced	**Lock position**, **Lock configuration**, and the read-only serialized definition. **Lock configuration** holds what the component shows, reads, looks like and does: the sections from **General** to **Visibility** are greyed out under a note saying so, and no panel or drag can rebind or restyle it. It can still be moved (unless **Lock position** is ticked too), put on another layer or grouped. Untick it to edit the component again.

Every text field leads with a **Key** box naming an entry in the project's string table, with the per-language boxes folded underneath. Under the key the inspector shows what it resolves to and which languages are still missing (*missing DE, FR, FUR, JA, KO, RU, ZH*). See *Strings* below.

Emptying a number box clears the property rather than storing zero, so a gauge scale of *nothing set* stays different from a scale of *zero*.

Binding signals and choosing an update rate

A binding says where a component's value comes from. Each slot a component declares gets a row showing its name, whether it is **REQUIRED**, and the data it accepts (**Numeric**, **Plottable** and so on). A required slot left empty is reported at validation.

The usual way to bind is from the **Signals** tab: select the component and drag a signal onto the canvas (it fills the component's first empty slot), or choose the slot in the tab's slot list and double-click the signal (see *The left panel*). The **source** list in the slot covers everything that cannot be dragged, and offers only what the slot accepts:

Source	Use it for
`Signal`, `VirtualSignal`	A signal from the catalogue (the ordinary case)
`Asset`	A field of a piece of equipment
`CurrentExecution`	The order, batch or run open on an asset now
`CurrentState`	The state an asset currently holds
`EventQuery`, `AlarmQuery`	Events or alarms, optionally for one asset and above a severity
`ContextValue`	Product, recipe, crew, tooling
`RelationshipQuery`	What feeds or contains an asset
`HistoricalStatistic`	A statistic over a window: `Last`, `First`, `Minimum`, `Maximum`, `Average`, `StandardDeviation`, `Count`, `Delta`
`Expression`	A number computed from the component's other slots
`Constant`	A literal, such as a fixed target
`PageParameter`	One of the page's parameters, so one page serves many machines

All binding sources **observe**; none writes. Pickers read the node's catalogue, so a name is picked rather than typed.

Choosing a signal for a slot. The **Signal** field opens *Choose a signal…*: the same list as the **Signals** panel — its search, its **Group by path**, its expanders and its *N of M* — in a wide dialog. Double-click a signal, or select it and press **Choose**; `Enter` chooses the selected one. The list's width can be changed by its right edge and is kept, per browser, for the next time. The field shows a long path shortened in the middle, with the whole path in its tooltip. The same list is used by the visibility rule, the interaction editor, a 3D model's hotspots and an automation step's signal setting.

What a reading is called. A trace in a legend, a cell of a value grid, a row of a list or a key-value table and a deviation bar are called, in this order: the binding's own **label**, when the author gave one (a key in the project's string table is shown in the viewer's language); the signal's **display name** from the project (see *Signals ▸ Display names*); and otherwise the last part of its topic. So a grid of the packaging line's machine rates needs no labels to read *Pouch loader rate*, *Case erector rate*; a label is for wording that belongs to this page — the overview calls the loader's throughput *Line rate* — or that must be translated:

a display name is one language, so the packaging demonstration, which speaks English, Spanish and Italian, keeps its translated labels. A binding that no longer resolves says *The catalog does not have this. It may have been renamed — pick the new one, or clear it.* A binding to a signal the node's activated revision no longer has — its connection was removed, say — says instead *The node's activated revision no longer has this signal. History still holds it; nothing live will arrive. Pick the signal that replaced it, or clear it.*, and **Validate** reports it as a warning, '*{slot}*' *binds* '*{signal}*', *which the node's activated revision no longer has*, the way the node's own configuration check warns about a computed signal that reads one.

A bound slot has two folded sub-sections:

- **Update and staleness**: **Update interval (ms)**, **Stale after, in seconds**, and a **Unit override** for this slot.
- **How it is drawn**: the trace's colour and line style (solid, dashed, dotted, dash-dot) and width. A trace that cannot be told apart by colour can always be told apart by pattern.

Update classes

Each component type has an **update class**, which is what most affects how a page performs. The **Page cost** tab groups a page's subscriptions by class.

Class	Interval	Used by
On change	When the value changes	States, indicators, alarms, text
Fast	About 250 ms	KPIs, numeric values, gauges
Moderate	About 500 ms	Mini trends
Detailed	About 100 ms, every sample	Live scopes and loggers (the lossless path)
Not live	—	Labels and decoration

When two components request the same signal at different rates, the fastest wins. Keep **Detailed** to the widgets whose value *is* the waveform; a whole page of scopes is an expensive page. Use a binding's **Update interval (ms)** to slow down a component that does not need its class's rate.

Thresholds

A threshold gives a value an operational meaning. Each band has an operator (`LessThan`, `LessThanOrEqual`, `GreaterThan`, `GreaterThanOrEqual`, `Between`, `Outside`, `Equals`, `NotEquals`), a value (and an upper value for `Between` and `Outside`), a semantic colour, a **label**, and **Make the component prominent in this band**.

Bands are *Evaluated top to bottom; the first match wins. Put the most severe band first.* The label is required: colour alone is not an acceptable way to convey state. The label box takes either a word or a string key. **Add a band** appends one; **Move up** reorders.

Visibility, navigation and interactions

Visibility holds one declarative condition and a **Hide when this is true, instead of showing** switch: `Always`, `HasRole`, `SignalCompare`, `SignalTrue`, `QualityGood`, `ExecutionOpen`, `StateIs`, `AlarmCountAbove`, `ParameterEquals`, `ParameterPresent`, `BreakpointAtLeast`. `HasRole` hides a control from somebody who could not use it; it is not a security boundary, since the node checks permissions on every request anyway. A component marked **Critical** takes no visibility rule.

Navigation sets what pressing the component does, one action per component:

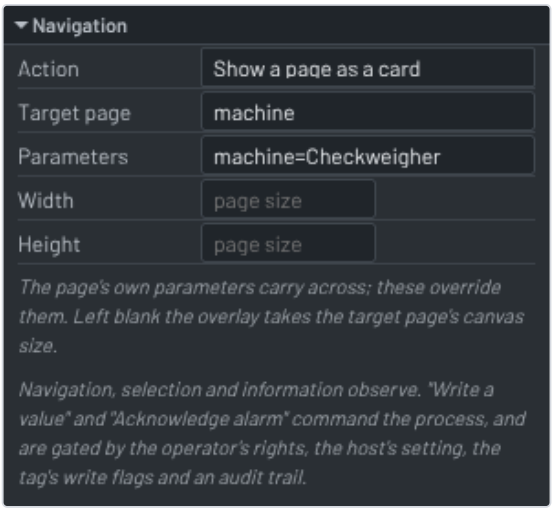
Action	Does
`NavigateToPage`, `NavigateToAsset`	Goes to another page, optionally in an asset's context
`OpenCard`	Shows another page as a small card beside the pointer. The one a mimic most needs.
`OpenDialog`	Shows another page centred over this one
`OpenTrend`	Opens chosen signals in a trend
`OpenAlarmDetails`, `OpenExecutionDetails`	Opens the detail
`SelectAsset`, `SelectArea`, `SelectTimeRange`, `SetPageParameter`	Changes the page's context
`OpenInformation`	Shows an information panel
`WriteTag`	Writes a value to a PLC tag: connection, tag, optional fixed value, and whether to confirm
`AcknowledgeAlarm`	Acknowledges the alarm the component is bound to

Signals of a linked node. Where this host reads other plants (Administration ▸ Linked nodes), their signals are in the **Signals** rail as a branch per plant, marked **read-only**, with ids like `Works:Reference/Treatment/TurbidityOut`. They bind to a value, a gauge or a trend like any other signal. They cannot be written: a `WriteTag` whose connection or tag belongs to a linked node shows "This tag belongs to Reference works, which this host only reads..." in the inspector, and the host refuses the write if an operator presses it. Writes go to the signal's own node, from that node's own screens.

`WriteTag` and `AcknowledgeAlarm` are the only constructs in a page that command. Adding one is a deliberate act by a person, and it grants nothing: the tag must be writable, its connection must allow writes, the host must have commands enabled, the operator needs the command permission, and every attempt passes the node's audited write path. Leaving the value blank makes the control an operator setpoint. On the demonstration **Command** page, **Order target** and **Target rate** are `WriteTag` controls with confirmation required.

The interaction editor, field by field

Open the component's **Navigation** section in the inspector. Choose the action in **Action**; the fields under it change with the action. Changing the action clears the fields of the previous one, so a page never carries a target page on an action that opens a trend.



The Navigation section of the Checkweigher block on the Line mimic page: Action Show a page as a card, Target page machine, Parameters machine=Checkweigher, and Width and Height left at page size

Action (as the list shows it)	Kind	Fields
— nothing —	`None`	None. Pressing the component does nothing.
Open a page	`NavigateToPage`	**Target page**, **Parameters**. The page opens with the parameters in effect and the listed ones on top, as a card does, and the listed ones are written in the address.
Show a page as a card	`OpenCard`	**Target page**, **Parameters**, **Width**, **Height**.
Show a page as a dialog	`OpenDialog`	**Target page**, **Parameters**, **Width**, **Height**.
Open an asset, **Select an asset**, **Select an area**	`NavigateToAsset`, `SelectAsset`, `SelectArea`	**Asset path**.
Open a trend	`OpenTrend`	**Signals** and **Range**.
Open alarm detail, **Open execution detail**	`OpenAlarmDetails`, `OpenExecutionDetails`	None.
Set the time range	`SelectTimeRange`	**Range**.
Set a page parameter	`SetPageParameter`	**Parameters**.
Show information	`OpenInformation`	**Text**.
Write a value (command)	`WriteTag`	**Connection**, **Tag**, **Value**, **Confirm before writing**.
Acknowledge alarm (command)	`AcknowledgeAlarm`	None.

What each field takes:

Field	Meaning
Target page	The page id to show, typed (`machines` , not *Machines*).

Field	Meaning
Parameters	Page parameters to pass, as `name=value` pairs separated by commas: `asset={asset}, area=Line1`. A value may name one of this page's own parameters in braces. For a card or a dialog, the target page also receives this page's parameters; the ones you list here override them. See *Page parameters*.
Width, **Height**	The size of a card or dialog, in pixels, in steps of 10. Left blank (*page size*), the overlay takes the target page's canvas size, kept within what the screen can show.
Asset path	An asset, picked from the node's asset tree (*Choose an asset...*). The list is empty where the deployment has no semantic model yet.
Signals	The signals the trend opens with, one numbered row each with a remove button. Add one with the **Signal** picker (*Add a signal...*); a signal already in the list is not added twice.
Range	A duration such as `15m`, `1h` or `8h`. For **Open a trend** it is how much history the trend opens on.
Text	The text of the information panel, typed in the box.
Connection	The id of the connection to write through.
Tag	The id of the tag to write.
Value	A fixed value to write each time (*fixed value — leave blank for an operator setpoint*). Blank makes the control a setpoint the operator enters.
Confirm before writing	On for a new control. The operator confirms each write before it is sent.

Under the fields the editor states what is still missing, in the same words validation uses:

Message	Means
A page navigation names no target page.	**Open a page** with no **Target page**.
An '{action}' interaction names no page to show.	A card or dialog with no **Target page**.
An '{action}' interaction names no asset.	One of the asset actions with no **Asset path**.
A trend drill-down names no signals.	**Open a trend** with an empty **Signals** list.
A time-range interaction names no range.	**Set the time range** with no **Range**.
A set-parameter interaction sets nothing.	**Set a page parameter** with no **Parameters**.
An information interaction has no text to show.	**Show information** with no **Text**.
A write command names no connection or tag to write to.	**Write a value** without **Connection** or **Tag**.
'*{range}*' *is not a duration.*	A **Range** that cannot be read, for example `1 hour` instead of `1h`.

Worked example: a trend from the checkweigher. On the demonstration **Overview** page, select the case weight gauge from the worked example at the end of this chapter, and open **Navigation**:

1. Set **Action** to **Open a trend**.
2. In the **Signal** picker, choose `Packaging/Checkweigher/CaseWeight`, then `Packaging/Checkweigher/WeightTarget`. They appear as rows 1 and 2.
3. Set **Range** to `1h`. The warning *A trend drill-down names no signals.* has gone.
4. Press **Save draft**. Once the page is published, pressing the gauge on an operator screen opens both signals in a trend over the last hour. **Preview** shows the page but does not carry out what pressing a component does.

Animation

On a process canvas a shape becomes an instrument through its animations. Each animation drives one visual property from one value. A shape's animations are in the inspector's **Driven by ({n})** section, which stands where **Data** stands on other components. A **Machine model (3D)** reads animations too (see *The machine model* below). No other component does; where one carries animations anyway, its section says *Only the drawn shapes and the machine model read animations.*

A shape with no animation says *Nothing drives this shape yet, so it is a drawing. Animate its fill to give it a state, or its level to make it a vessel.*

Adding, opening and removing an animation

1. In **What the new animation drives**, choose what to drive. The list offers only what this shape can show.
2. Press **Animate**. The animation is added and opened.
3. Fill in its source and its range or bands, as below.

Each animation is one row, folded: what it drives, then what drives it (a signal path, or — *not bound* —). The pencil (**Edit**) opens the row and closes any other; **Remove** deletes the animation. Opened, the row shows:

Field	Shown for	Meaning
Drives	Every animation	What the animation drives. Changing it resets **Gives** and **through** to the new target's defaults.
Driven by	Every animation	The source, edited like any binding slot (see *Binding signals*): a signal, an expression, a page parameter and so on. For a signal, choose **Signal** in the source list and pick it in **Signal** (*Choose a signal...*).
From value, **To value**	Range targets	The input range: the values that map onto the two ends of the output. New animations start at 0 and 100.
Gives (with its unit), **through**	Range targets	The output range the input maps onto.
Range	Range targets	**Hold at the ends when the value goes outside the range**, on by default. Under it a sentence reads the mapping back, for example *0 → 0, 100 → 1*, with *extrapolated beyond* added when the switch is off.
Bands	Band targets	A threshold editor, as in *Thresholds*: each band an operator, a value, a colour and a label. A new band animation starts with one band, *On* (not equal to 0, Running colour).

What a drawing can drive:

In the list	Kind	How it is driven	Default output	Applies to
Fill colour	`Fill`	Bands	—	Closed shapes, text and icons
Outline colour	`Stroke`	Bands	—	Every shape but an image
Outline width	`StrokeWidth`	Range	1 to 6 units	Every shape but an image
Opacity	`Opacity`	Range	0 to 1	Every shape
Rotation	`Rotate`	Range	0 to 360°	Every shape

In the list	Kind	How it is driven	Default output	Applies to
Move across	`TranslateX`	Range	0 to 100 units	Every shape
Move down	`TranslateY`	Range	0 to 100 units	Every shape
Stretch across	`ScaleX`	Range	0 to 1×	Every shape
Stretch down	`ScaleY`	Range	0 to 1×	Every shape
Fill level	`FillLevel`	Range	0 to 1, filled from the bottom	Closed shapes: rectangle, ellipse, polygon, arc, symbol
Text	`Text`	The value itself, formatted by **Formatting**	—	Text shapes
Blink	`Blink`	Bands: the shape flashes while the value is in a band	—	Every shape
Flow along the run	`Flow`	Range	0 to 40 units per second	Lines and runs
Keep turning	`Spin`	Range	0 to 90 rpm	Every shape
State word	`StateText`	Bands: the matching band's label becomes the text	—	Text shapes

A few rules worth knowing:

- **Colour is never alone.** A band needs a label, and a shape coloured by a band shows the band's word unless you turn off **Show the band label** in the shape's own settings.
- **Flow stops at zero.** A value of zero stops the movement, and a negative value runs it backwards, so a conveyor or pipe that is not moving does not look as if it is.
- **A bad value does not move the drawing.** On a shape, every animation leaves the shape as drawn when the value behind it cannot be trusted.
- Under an opened row, the editor states what is wrong with it, naming the animation by its number. For example: *… has no bands, so it never changes anything.*, *… declares an empty input range, so every value maps to the same appearance.*, *… maps a fill level outside 0…1.* or *… drives nothing: it has no source.*

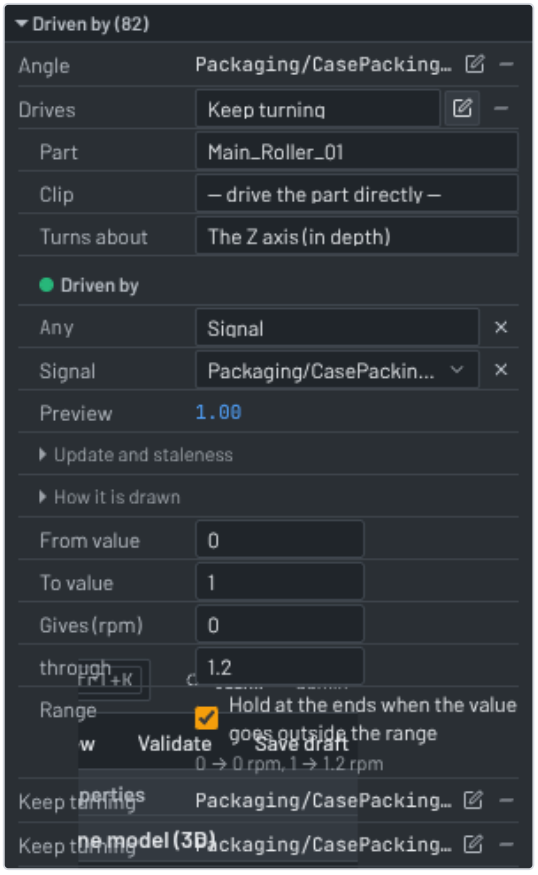
The machine model

A **Machine model (3D)** has no binding slots: animations are the only way to drive it. Choose its file in **Model**; uploading a model is described in *Assets: images and 3D models*. A machine offers six targets, in a machine's words:

In the list	Kind	What it does
Keep turning	`Spin`	Turns the part continuously, at a rate the value sets.
Angle	`Rotate`	Puts the part at an angle.
Slide	`TranslateX`	Moves the part across.
Lift	`TranslateY`	Moves the part up.
Move in depth	`TranslateZ`	Moves the part away from or towards the viewer.
Part colour	`Fill`	Colours the part from the band the value falls in.

An opened machine animation has three more fields above **Driven by**:

Field	Meaning
Part	The part to drive, by the name it carries in the model file. The Studio reads the names from the model and offers them in a list (— *pick a part* —). If the model cannot be read, it is a text box (*the part's name in the file*). For **Part colour** the empty choice is *The whole machine*.
Clip	Shown when the model carries animation clips. Choose a clip to have the value set its playback speed, instead of driving the part directly (— *drive the part directly* —). Where a model has clips, this is usually the better choice: the clip's articulation was made once by whoever animated the model.
Turns about	For **Keep turning** and **Angle** only. **The X axis (across)**, **The Y axis (upright)** (the default) or **The Z axis (in depth)**.



The Driven by section of the Line in 3D machine model with one Keep turning animation opened: Part Main_Roller_01, Clip drive the part directly, Turns about The Z axis (in depth), driven by a Packaging/CasePacking signal, From value 0 to 1 giving 0 to 1.2 rpm

A newly placed machine model shows **Driven by (0)** with the note *Nothing drives this machine yet*; press **Animate** there to add its first animation. Part names are checked against the file when you publish, so a part renamed in CAD is refused then rather than silently doing nothing.

Part appearance finishes each part as the machine really looks, before any tag touches it. **Give a part its own appearance** adds a row:

Field	Meaning
Part	A part of the model, or *The whole machine*.
Paint	A paint from the theme's list (— *leave it as it is* —, *Surface*, *Outline*, *Text*, or a status colour such as *Attention*).

Field	Meaning
Texture	An uploaded image wrapped onto the part (*no texture*; *no images uploaded yet* when the project has none).
Tiles	How many times the texture repeats across the part. At least 0.1; 1 by default.



The Appearance section of the Line in 3D machine model: one Part appearance row, Safety_Aisles painted Attention with no texture (no images uploaded yet) and Tiles 1, then Show the band label, Turn 34, Tilt 26, Zoom 1.45, Let the viewer turn it and Cast shadows

A **Part colour** animation paints over the appearance while its band matches, and the part returns to its appearance when the band clears. A part nothing names keeps the materials the file gave it. A row with neither a paint nor a texture is reported: *sets neither a texture nor a paint, so it does nothing*.

Part detail lets pressing a part open the detail of the signals that describe it, without leaving the page. **Let a part open its own detail** adds a row:

Field	Meaning
Part	The part an operator presses. Required: a row with no part is reported as *names no part, so there is nothing to press.*
Called	What to call the part, as a key or per-language text. Left empty, the part's name in the file is used.
Opens on	How much history the detail opens with, as a duration such as `1h`. Empty takes the runtime's default.
Signals	The signals the detail charts, in order, one numbered row each with a remove button, added with *Add a signal…*. Required.

Each part with a detail also appears as a named button on the machine, so that a keyboard or screen-reader user can reach it. What the operator sees is described in *The browser operator HMI* chapter. Without any row the section says *Pressing the machine does nothing. Add a part and the signals that describe it, and pressing that part opens their charts without leaving the page.*

On the demonstration. The 3D packaging cell (page `line3d`, **Line in 3D**) is built this way:

- **Part appearance**: the part `Safety_Aisles` is painted *Attention*, the colour of the walkways.

- **Part colour** on `M01_Magazine_Case_Stack`, driven by `Packaging/Erector/MagazineLevel`, with one band: **Comparison** `<` (*less than*) 25, **Colour** *Warning*, and the label key `magazineLow` (*Magazine low*).
- **Keep turning** on `M10_Wrapper_Rotary_Arm` about **The Y axis (upright)**, driven by `Packaging/CasePacking/WrapperActive`.
- **Part detail** on each of the eleven machines. For the checkweigher: **Part** `M04_Case_Checkweigher`, **Called** with the key `checkweigher`, **Opens on** `1h`, and the signals `Packaging/Checkweigher/CaseWeight`, `Packaging/Checkweigher/WeightTarget` and `Packaging/Checkweigher/UnderweightCount`.

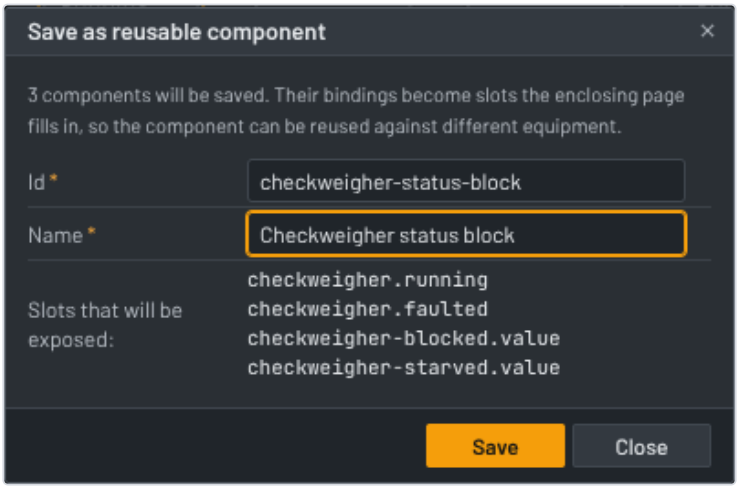
Composites and templates

Save as reusable component turns the selection into a **composite**. A composite placed on a page is an instance: change the definition and every instance follows, so a valve detail card drawn once serves fifty valves. **Save as template** saves a whole page, which then appears under **Start from** in **New page** beside the shipped templates and **Blank**. A page bound through a page parameter becomes one screen per machine: the demonstration **Machine** page (`machine`) binds `Virtual/Computed/{machine}Throughput` and `Packaging/{machine}/RatedSpeed`, and the `{machine}` parameter is supplied by whatever opens it. On the **Line mimic**, each machine opens it as a card with **Parameters** `machine=Checkweigher`, `machine=Loader` and so on.

The Save as reusable component dialog

Select the components, then press **Save as reusable component** on the toolbar. The dialog says how many components it will save, and that *their bindings become slots the enclosing page fills in, so the component can be reused against different equipment.*

Field	Meaning
Id	Required. The composite's id (for example `pump-status-block`). It follows the name until you edit it.
Name	Required. What the library calls it (for example *Pump status block*).
Slots that will be exposed:	Read-only. One line per bound slot in the selection, as `{component id}.{slot}`. These become the composite's binding slots, and the bindings themselves are not saved with it. *None of the selected components is bound, so the component will have no slots.* when there is nothing bound.



The Save as reusable component dialog for the Checkweigher block and its Blocked and Starved lamps on the Line mimic page: 3 components will be saved, Id checkweigher-status-block, Name Checkweigher status block, and the four slots that will be exposed

Press **Save**. The result reads *Saved as '{id}', version {n}.* and *It exposes {n} binding slots and can be inserted on any page in this project.* The positions inside the composite are kept relative to the selection's own corner, so it can be placed anywhere. If the save fails the dialog says *The component could not be saved. Check the id is not already in use.*

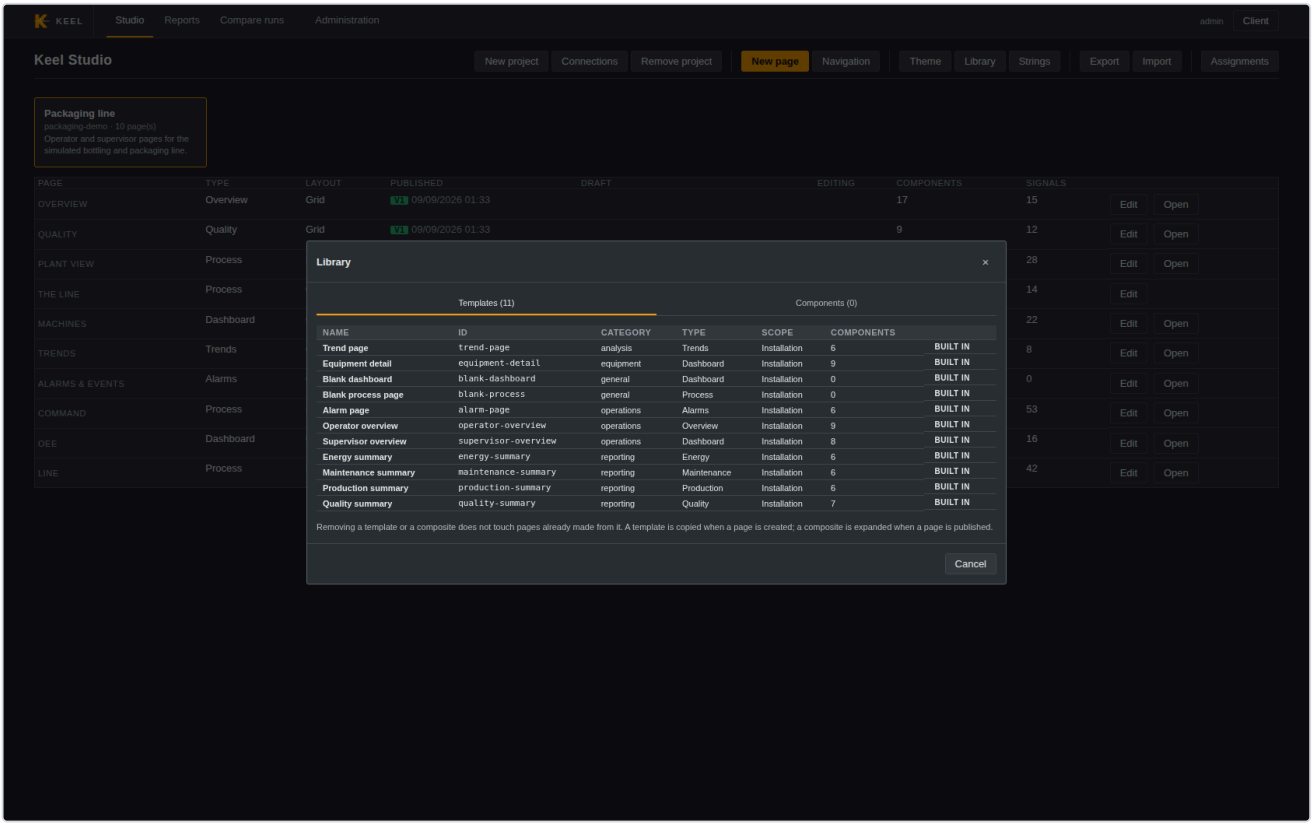
The Save as template dialog

Save as template is on the toolbar for somebody who may manage templates. It saves the whole page; *The page is copied as it stands, bindings included. Component ids are regenerated each time a page is made from it, so two pages from one template never share ids.*

Field	Meaning
Id	Required. Starts as the page name made into an id; it follows **Name** until you type an id yourself.
Name	Required. Starts as the page's name.
Category	Free text. Starts as the page's type in lower case, for example `process`. A template in the category `general` is offered in **New page** for every page type; any other category is offered only for pages of the template's own type.
Scope	**Available to every project**. Off by default: the template belongs to this project. On, it is shared across the installation.

Press **Create**. The result reads *Saved as '{id}'.* and either *Pages in this project can start from it.* or *Every project can start a page from it.*

The library



The Library: the eleven built-in templates with their category, type, scope and component count

Library… lists what exists in two tabs, each with a count.

Tab	Columns
Templates	**Name**, **Id**, **Category**, **Type**, **Scope**, **Components**, **Protection**
Components (the composites)	**Name**, **Id**, **Category**, **Scope**, **Version**, **Slots**, **Components**, **Protection**

Scope is **This project** or **Installation** (shared by every project). **Protection** shows **Built in** for a template shipped with the product and **Locked** for one that cannot be changed here. The eleven built-in templates are **Blank dashboard**, **Blank process page**, **Operator overview**, **Supervisor overview**, **Equipment detail**, **Alarm page**, **Trend page**, **Production summary**, **Quality summary**, **Energy summary** and **Maintenance summary**.

Each row has a menu (right click, `Menu` key, or ⋯) with **Delete**. It is unavailable, and says why, for a built-in template (**Built in**), a locked entry (**Locked**), or when you may not manage templates or composites (*You are not permitted to change the library.*). Delete asks first: *Delete “{name}”? No author on this installation will be able to use it again.*

What deleting does to pages already made:

- **A template** is copied when a page is created from it, so deleting it leaves those pages as they are.
- **A composite** is drawn from its definition each time a page is shown; that is how changing the definition updates every instance. Deleting a composite takes it out of the library and keeps its definition for the pages that already use it: their instances go on drawing it, on the canvas, in **Preview** and on the operator screens, and validation does not report them. Nobody can pick it from the library again. Saving a reusable component under the same id puts it back in the library, and every instance follows the new definition.

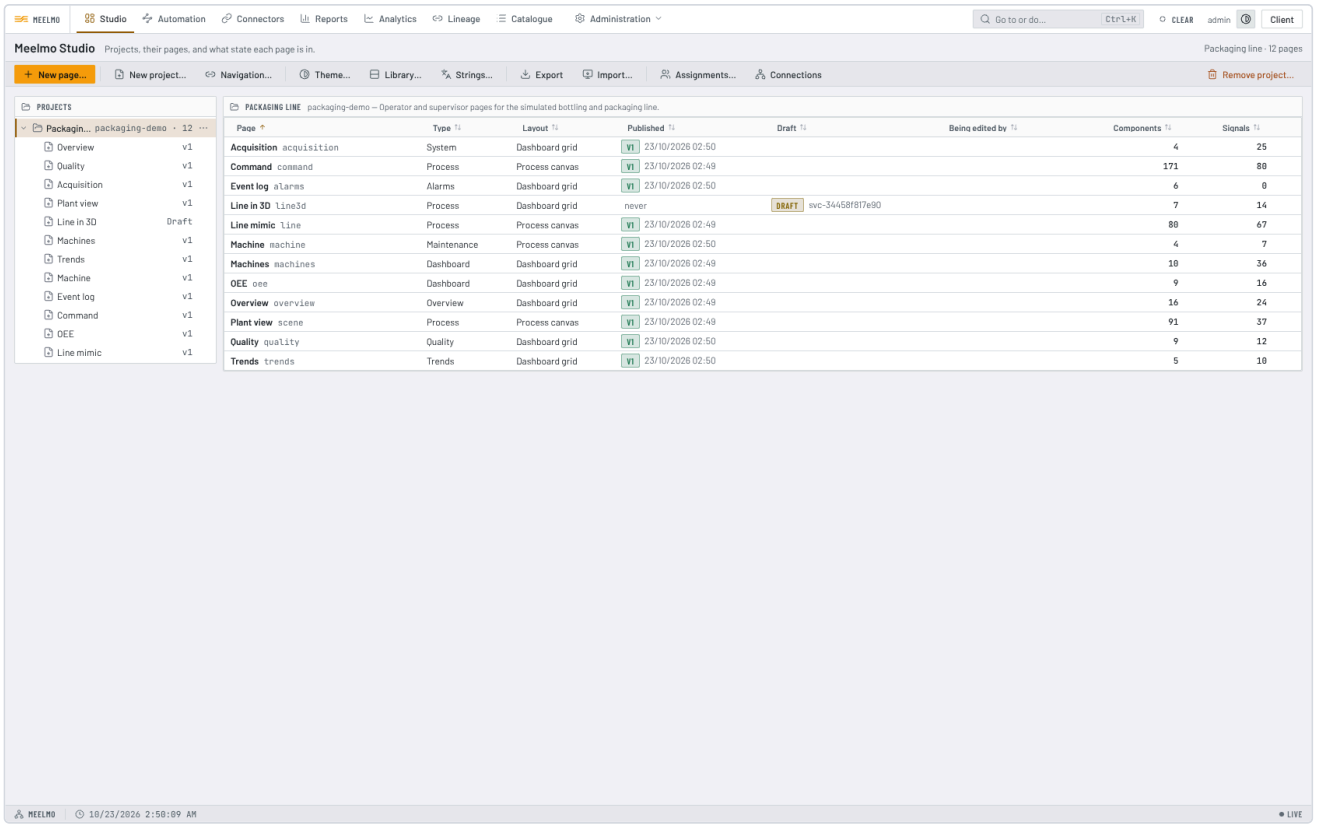
Strings

Every authored text (titles, labels, page names, navigation labels, band labels) can either be typed per language on the component, or name a **key** in the project's string table. **Strings…** edits the table: each row is a key and its text in every language. Translate *Filler* once and every screen naming that key follows, including screens already published, without republishing.

The Strings screen refuses to save two rows under one key, a row with no English, and text with no key. In the inspector, a key the table does not answer is shown as the key itself and reported by validation at *Warning*. The demonstration packaging project keys every piece of text: 108 distinct wordings behind 334 pieces of text on its screens.

Theme

Theme… opens *Theme and palette*, which sets the project's colours, type and shell layout; every page in the project changes together.



The theme editor: base theme, status colours, surfaces and text, trend series and typography

Setting	What it does
Base theme	*Industrial dark* (default, drawn in the control-room style described below), *Industrial light*, *Console navy*, *Control amber*, *Slate light*, *Midnight*, *High contrast*, *Phosphor* or *Meelmo*. Each says what it is for, for example *A mimic wall seen across a darkened control room.* for Console navy. Pick for the room the screen is in.
Operators may switch between light and dark	Whether a viewer may override the theme from the ⋯ menu.
Status colours	The ten colours the runtime uses to mean something: **Running**, **Ready**, **Attention**, **Warning**, **Alarm**, **Critical**, **Info**, **Unavailable**, **Stale**, **Bad quality**. Change one here and every screen follows.
Surfaces and text	**Background**, **Surface**, **Raised**, **Sunken**, **Text**, **Dim text**, **Border**, **Grid line**.
Trend series	**Series 1** to **Series 4**, the colours a chart cycles through.
Typography	The **Interface font** (*Theme default*, *System*, *Arial*, *Verdana*, *Tahoma*, *Georgia — serif* or *Monospace*) and five sizes in pixels: **Base text**, **Small text**, **Readout**, **Large readout**, **Hero readout**.
Density and shape	**Corner radius**, **Border width**, and four spacings: **Spacing · tight**, **Spacing · snug**, **Spacing · roomy**, **Spacing · loose**.

The control-room style. Every theme draws its pages the same way, and differs only in its colours. A panel sits on the page's ground with no border and the ground showing round it; readings inside a panel are tiles one tone darker. On a tile the label is small, spaced capitals in the muted ink, the reading is the largest thing on it in the full ink, and the unit is set well under half the reading's size in the dim ink. Labels and readings share one condensed face (Barlow Condensed, served by the node itself) with every figure the same width, so a column of readings lines up and a reading that ticks over does not shuffle its unit; tag names, addresses and expressions keep the monospaced face. A status pill is outlined and lettered in

its status colour on no fill of its own, and colour is kept for status: a badge that only describes its tile — a chart's window — is grey. A chart or a list with nothing to show says so in one line of the label's voice — *No data for this window*, *No active alarms*, *No sample* — rather than drawing an empty frame.

A **Preview** sample (a *Sealer temperature* card and chips in the Running, Warning, Alarm and Critical colours) is painted from the values as you change them. Each field has ↺ (*Reset {field} to the base theme*); **Reset all** returns everything to the base theme; **Save theme** writes the theme and the layout to the project; **Cancel** (or `Esc`) closes without saving. A problem that stops the save is shown at the top of the dialog.

Layout

The **Layout** block decides which bands the runtime draws around a page. A wireframe of the shell beside the fields moves as you change them.

Field	Default	What it does at runtime
Show header	On	Off, the header starts closed rather than going away: the runtime keeps a tab on the top edge that opens it, because the header carries the site selector, the language, sign-out and the way home.
Show navigation rail	On	Off, the page rail is not drawn. The **Navigation** buttons are disabled while it is off.
Show footer	On	Off, no footer is drawn, and the controls that live in the footer (home, the replay control and the ⋯ menu) are drawn in the header instead.
Prefer full screen (kiosk)	Off	On, the runtime asks the browser to go full screen when a page first opens. Browsers grant that only to an installed app or a kiosk browser; in an ordinary tab the request is ignored until the operator chooses full screen.
Navigation	**Top**	Where the rail sits: **Top**, **Left**, **Right**, **Bottom** or **Overlay**. Top and bottom run as a band; left and right stand as a column. **Overlay** collapses the rail and opens it over the plant as a drawer. A phone always uses the overlay, whatever this says.
Alarm banner	**In header**	Where the worst-alarm banner sits: **In header**, **Top band**, **Bottom band** or **None**. **None** is for installations with no alarms configured. It hides only the shell's strip: no alarm is silenced, acknowledged or hidden from the alarm pages, the header's counts or any alarm component.



The Layout block of Theme and palette set for a wall display: Show header and Show footer cleared, Show navigation rail and Prefer full screen (kiosk) ticked, Navigation on Overlay and Alarm banner on Top band, with the wireframe beside it showing the banner band and the rail as a drawer

Worked example: a wall display for the packaging line. The demonstration's **Command** page is the one a shift leader leaves up on a wall screen. To make that screen show the plant and little else:

1. In the project browser, select **Packaging line** and press **Theme…**.
2. Set **Base theme** to *Console navy*, for a screen seen across a darkened room, and untick **Operators may switch between light and dark**, so nobody walking past changes it.

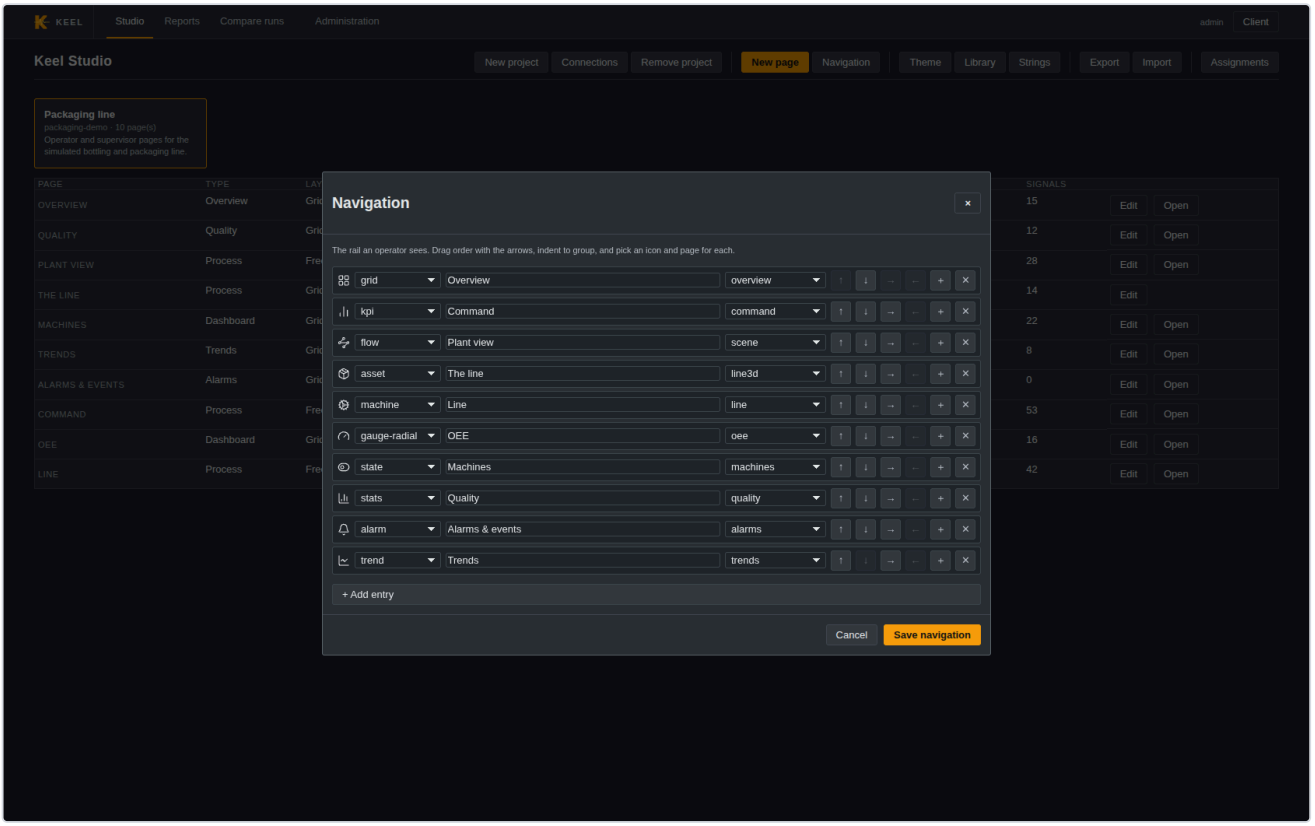
3. Under **Layout**, untick **Show header** and **Show footer**, tick **Prefer full screen (kiosk)**, and press **Overlay** under **Navigation**. The wireframe loses its header and footer bands, and draws the rail as a drawer over the page.
4. Leave **Alarm banner** on **Top band** or **In header**: this plant has alarms, and the quiet banner is what tells the room nothing is wrong.
5. Press **Save theme**. Every page of the project takes the new layout. If only one screen should look like this, give it its own project and assign that project to its station (see *Assignments*).

Navigation

Navigation… builds the rail operators see: *The rail an operator sees. Order with the arrows, indent to group, and pick an icon and page for each.* The rail is shown as a tree on the left; the entry you select is edited on the right.

Field	Meaning
Label	What the operator reads on the rail. An entry with no label shows as *(no label)* in the tree.
Icon	The entry's icon, from the shared library (for example `grid`, `kpi`, `flow`, `machine`, `gauge-radial`). A rail across the top or bottom of the screen draws it beside the name in a slim tab; a rail down the left or right edge and the phone's drawer draw it over the name.
Page	The page id the entry opens. Leave it on — *group* — to make a heading that groups the entries under it rather than a link.

Add entry above the tree appends an entry at the top level. The buttons beside it act on the selected entry, and the same commands are on each entry's menu (right click, `Menu` key, or ⋯): **Add an entry inside**, **Move up**, **Move down**, **Indent under the entry above**, **Move out of its group**, and **Remove**. Nothing is written until you press **Save navigation**; **Cancel** discards the changes, removals included.



The Navigation editor on an earlier build, with one row per rail entry and arrow, add and remove buttons on each row

The figure shows an earlier build, where every row carried its own ↑ ↓ → ← + × buttons; those commands are now the buttons over the tree and the entry's menu. The demonstration rail runs **Overview**, **Command**, **Plant view**, **Line in 3D**, **Line mimic**, **OEE**, **Machines**, **Quality**, **Event log**, **Trends** and **Acquisition**; an entry whose page is not published is not shown to operators.

Layout and breakpoints

A page is designed at one breakpoint and adapts to the others:

Breakpoint	From	Columns
`Phone`	0 px	2
`TabletLandscape`	768 px	6
`Laptop`	1280 px	12
`FullHd`	1680 px	12 (the design target)
`LargeDisplay`	2400 px	12

Design at **Full HD**; the Studio arranges the others. Each breakpoint keeps its own placement, and the inspector's **Responsive** section marks the ones still *inherited*. A phone page scrolls rather than squeezing every row.

Automatic arrangement

Every other resolution is derived from the Full HD layout:

- **Grid pages** are laid into the narrower or wider grid in reading order — row by row, left to right — so nothing is placed ahead of what came before it. Each component keeps its share of the width (a half-page trend stays half the tablet and takes the whole phone), is widened to its card's minimum size so nothing is cut, and keeps its height in rows. Where the target has as many columns as Full HD (Laptop, Large display) the layout is copied as it is.
- **Process pages** are one drawing scaled to each screen, letterboxed so every element keeps its place relative to the others. Where that scale would draw the text or a component too small to read, the badge reads **Too small** and its tooltip says how small; operators will have to zoom there, and a page drawn for that screen is the better answer.

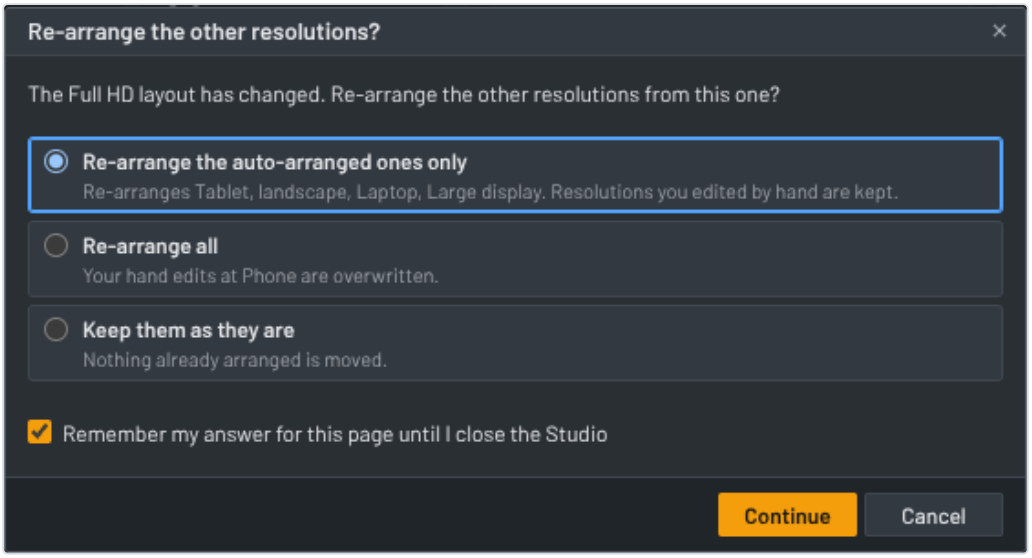
A new page is arranged at every resolution when it is created, and each resolution is marked **Auto**. Moving or resizing a component at a resolution by hand marks that resolution **Edited**.

When you save or publish after changing the Full HD layout, the Studio asks once: *Re-arrange the other resolutions?*

Answer	What happens
Re-arrange the auto-arranged ones only (the default)	The **Auto** resolutions follow the new Full HD layout; the **Edited** ones are kept.
Re-arrange all	Every resolution follows, and the dialog names the edited ones whose hand edits are overwritten.
Keep them as they are	Nothing already arranged moves.

Whatever you answer, a resolution with no arrangement, or a component with no place at a resolution, is filled: a page is never saved or published with a resolution missing. Leave **Remember my answer for this page until I close the Studio** ticked and later saves of the page apply the same answer without

asking.

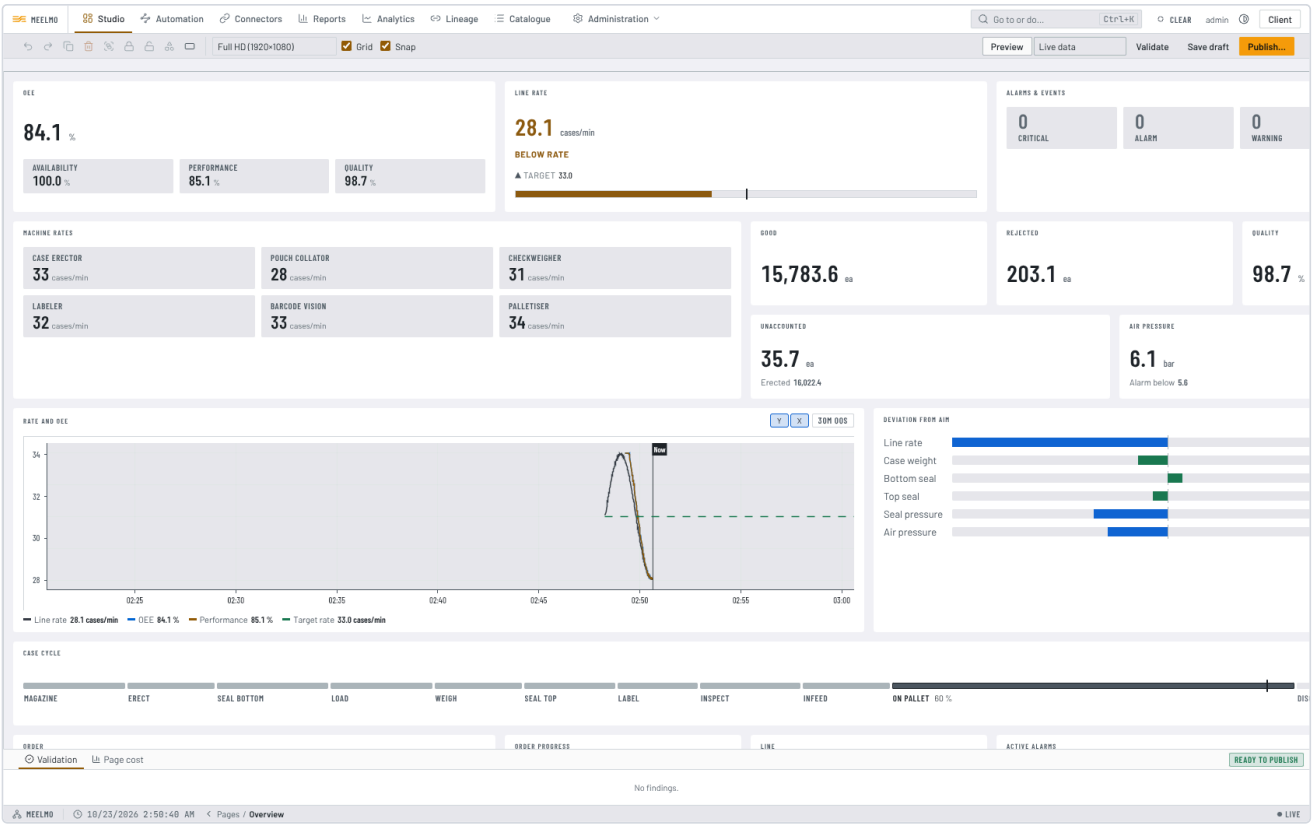


The question on saving after the Full HD layout changed: re-arrange the auto-arranged ones only, re-arrange all (overwriting the hand edits at Phone), or keep them

Pages saved by an earlier version carry no record of which resolutions were automatic, so every resolution they have is treated as **Edited** and nothing is overwritten unless you choose **Re-arrange all**. If such a page reports *At Phone this component spans to column 12 in a 2-column grid* or *At Phone this component overlaps …*, the component was added after the other resolutions were laid out and inherited its Full HD place there: press **Arrange missing resolutions** on the page, or in the project browser for the whole project.

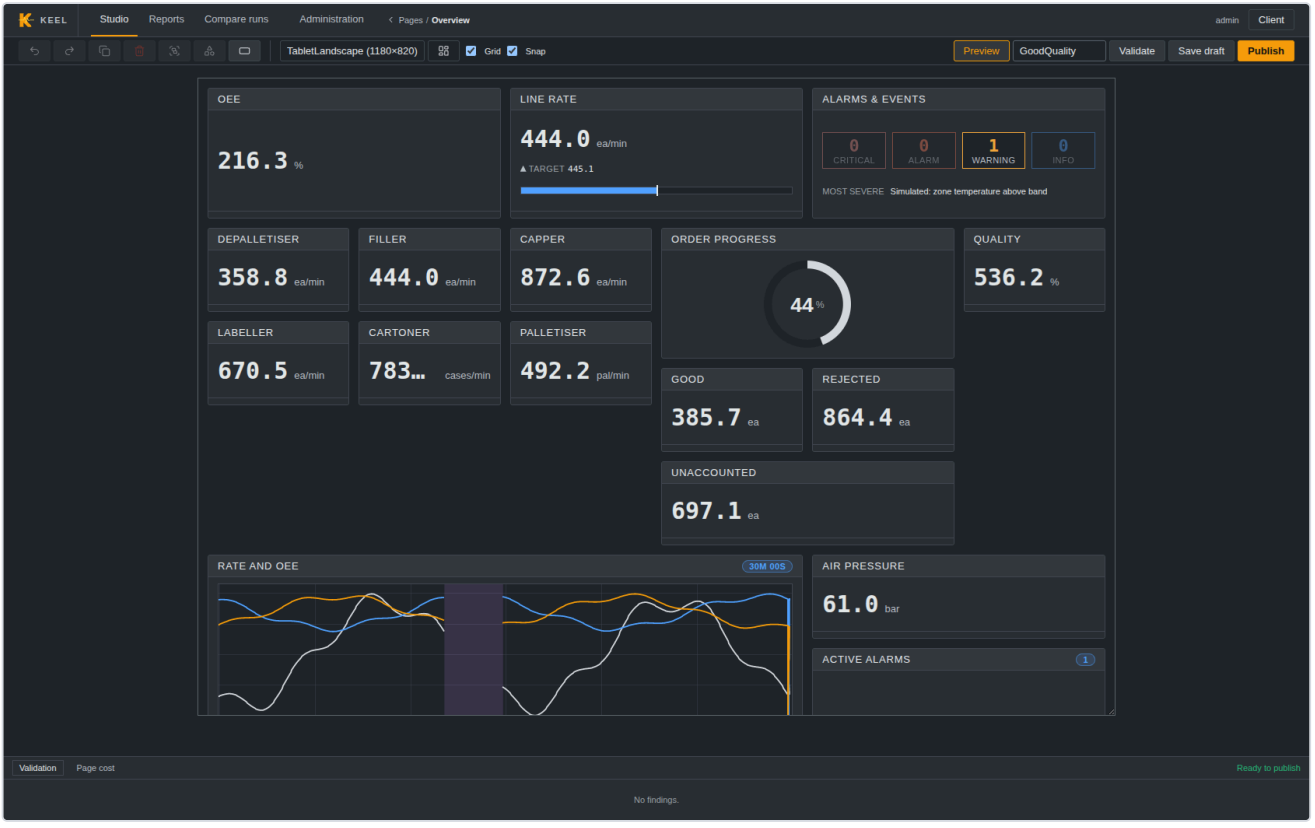
Preview and simulate

Preview replaces the canvas with the page as an operator receives it, at the exact nominal size of the chosen breakpoint, with no selection handles, grid or palette. It shows the page, not the shell: there is no rail and no assignment is applied. Preview does not change the draft.



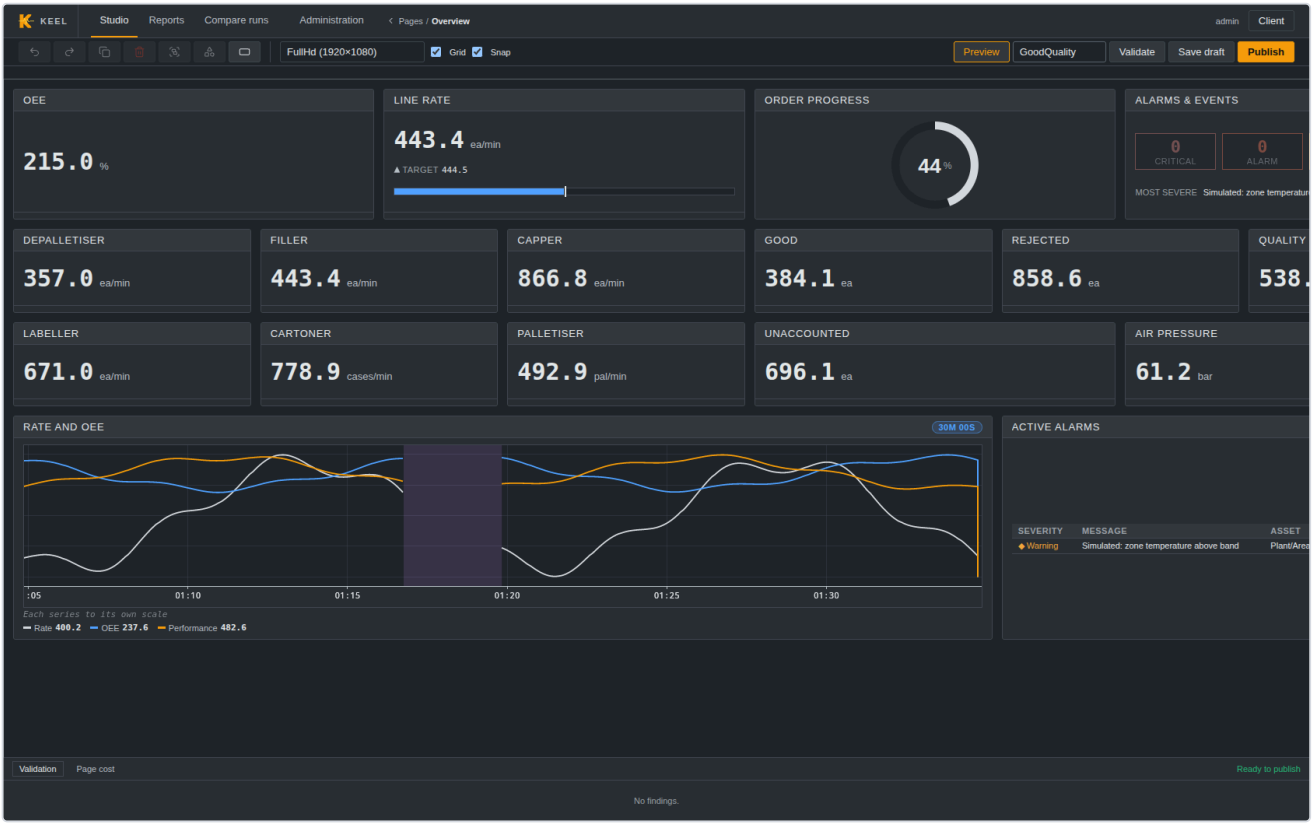
Preview: the Overview page drawn as an operator receives it

The breakpoint selector still works in preview, which is how you check a width you are not designing at:



The same page previewed at the tablet breakpoint

While previewing, a scenario selector appears beside **Preview**. **Live data** (the default) shows real readings; every other scenario drives the page from generated values, so you can check a threshold, an animation or a visibility rule without waiting for the plant to reach the condition. Alarms a scenario produces read *Simulated: …*, so a screenshot cannot be mistaken for the plant.

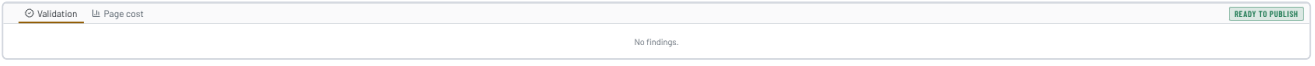


Simulating a scenario: the GoodQuality scenario selected beside Preview, and a simulated warning in the alarm list

Validation and page cost

The strip along the bottom has two tabs and, on the right, the page's publishability (**READY TO PUBLISH**, or what is stopping it).

Validation lists findings at **Error**, **Warning** and **Information**, each naming the component id it is about (for example `u-checkweigher`). A page with errors cannot be published. On the demonstration **Command** page, ten warnings such as '*Packaging/Checkweigher/RatedSpeed*' *is bound to both* '*target*' *and* '*maximum*' *on the same component* are advice, and the page is still ready to publish.



The validation strip

Page cost estimates what the page will demand once open: its subscription groups by update class (*{n} signals every {ms} ms*), high-frequency signals, the trend point budget, the estimated browser update rate per second, and whether it needs the semantic model. Read it before publishing a screen that will sit on a wall for a year. The usual causes of an expensive page are a trend with a large point budget, a semantic list over a wide window, and too many live components on one screen.

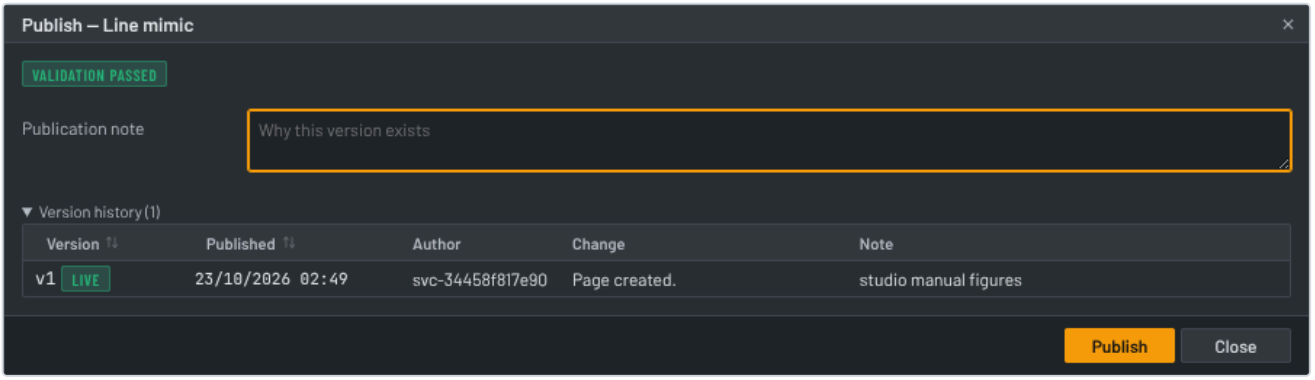
Editing at the same time as somebody else

A page being edited is held by a **lease**, renewed while you work. Another author opening it gets it read-only, with *Read-only — another user is editing this page*. Somebody with the break-lease permission can press **Take over editing**. The lease protects work from being silently overwritten; it is not a security feature.

Publishing, versions and rollback

The lifecycle is draft, validate, publish:

- **Save draft** keeps your work without changing what operators see.
- **Publish…** opens the publish dialog. It validates first (*Validation passed*, or *{n} errors must be fixed first*), asks for a **Publication note** (*Why this version exists*), and publishes. The result reads *Published as version {n}. Operators will see it the next time their client loads this page.*
- **Version history** in the same dialog lists every published version with its **Version**, **Author**, **Change** and **Note**, the live one marked *live*. **Roll back** activates an earlier version; a rollback is itself a publish, so the history shows what was live at any moment.

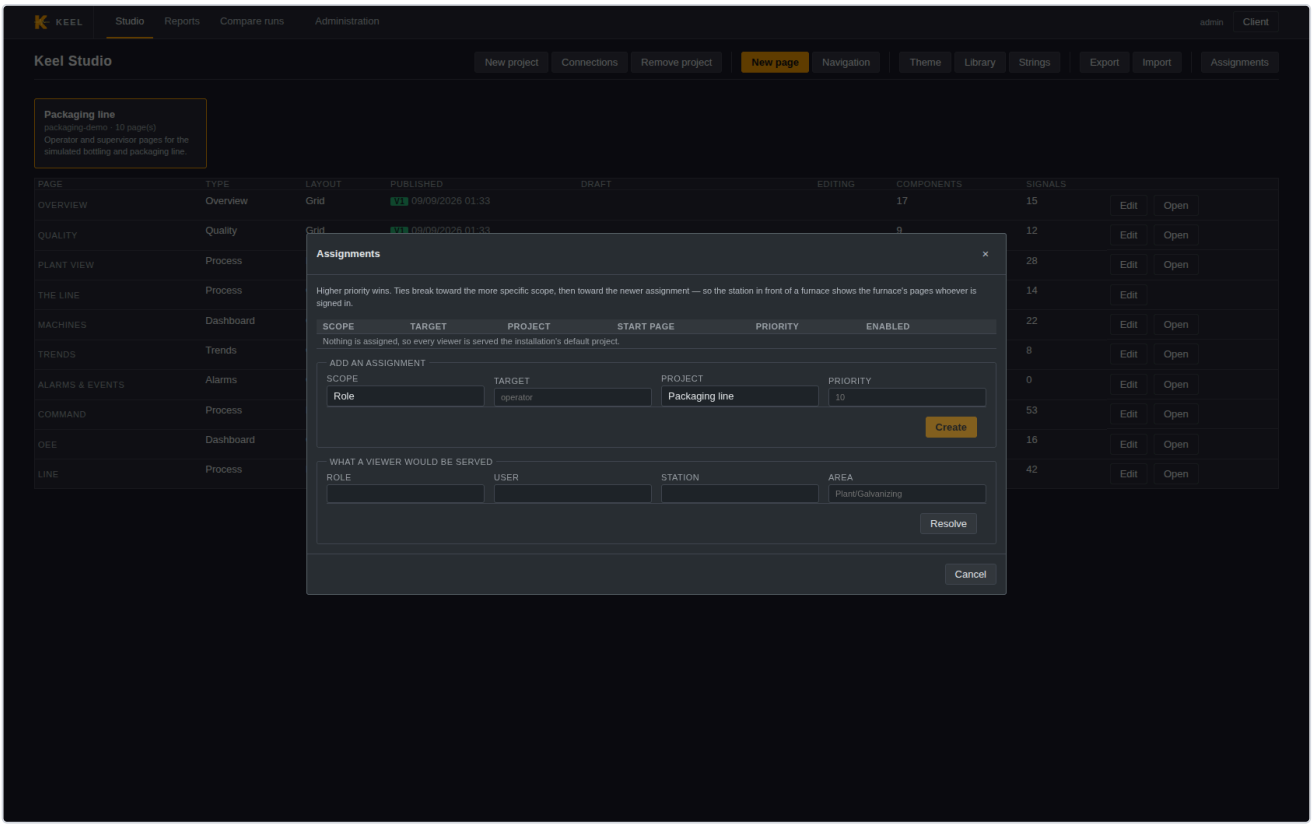


The publish dialog for the Line mimic page: VALIDATION PASSED, an empty Publication note box reading Why this version exists, and Version history with v1 live

Publishing can be refused with a reason: *There is nothing new to publish.*, *The page changed while you were editing it, so this draft was not published.*, *Somebody else is editing this page.*, or *You are not permitted to publish pages.*

Assignments

Assignments… decides which project, and optionally which start page, each viewer lands on. With nothing assigned, *every viewer is served the installation's default project*.



The Assignments dialog: the rule, the assignment table, Add an assignment with Scope, Target, Project and Priority, and What a viewer would be served with Resolve

Field	Meaning
Scope	**Everyone**, **Role**, **User**, **Station** or **Area**.
Target	The role, username, station profile or area path. Empty for Everyone.
Project	The project served.
Start page	Optional; the page the viewer lands on.
Priority	Higher wins. Ties break toward the more specific scope, then toward the newer assignment.
Enabled	An assignment can be switched off without deleting it.

Create adds the assignment. Underneath, **What a viewer would be served** takes a **Role**, **User**, **Station** and **Area**, and **Resolve** answers with the project and page that viewer would get. Use it before relying on an assignment set.

Worked example. To send the line supervisors to the OEE page:

1. Open **Assignments…**.
2. Under **Add an assignment**, set **Scope** to `Role`, **Target** to `LineSupervisor`, **Project** to `Packaging line`, **Start page** to `oee`, **Priority** to `10`.
3. Press **Create**.
4. Under **What a viewer would be served**, enter **Role** `LineSupervisor` and press **Resolve**. The answer is the Packaging line project, page `oee`.
5. Enter **Station** `line-1-pulpit` as well and press **Resolve** again. Where the station has its own assignment at priority 40, as in the demonstration, the station wins and the answer is the `line` page.

Worked example: a case weight gauge on the Overview page

The checkweigher's case weight, `Packaging/Checkweigher/CaseWeight`, should sit around its target `Packaging/Checkweigher/WeightTarget`; the demonstration's automation treats more than 1255 g held for 10 s as a giveaway problem. This example adds a gauge for it to the **Overview** page and publishes it.

1. Open **Studio**, select the **Packaging line** project and open the **Overview** page.
2. Make room: select the **Air pressure** tile and drag its right edge to free two columns in the fourth row (or add the gauge to an empty row below).
3. In the **Components** tab, type `gauge` in the search box and drag **Linear gauge** (under **Gauges and indicators**) into the free cells.
4. With the gauge selected, open the **Signals** tab and type `CaseWeight`. In the slot list at the bottom of the tab (*Linear gauge — Signals*), press **Actual value**, then double-click `Packaging/Checkweigher/CaseWeight`. The inspector's **Data** section shows the binding and its current value, for example `1,255 g`.
5. Press **Target value** in the same list and double-click `Packaging/Checkweigher/WeightTarget`, so the gauge draws a target mark.
6. In the gauge's own properties set **Scale minimum** to `1200` and **Scale maximum** to `1300`, and leave **Show the scale** and **Show the reading** on.
7. In **General**, set the **Title** key to an existing string such as `caseWeight`, or type `Case weight` in the English box.
8. In **Formatting**, set **Decimal places** to `0`; the unit `g` comes from the signal.
9. In **Thresholds**, press **Add a band** twice and fill in, in this order:
   - `GreaterThan` `1300`, colour *Alarm*, label `Sensor fault`;
   - `GreaterThan` `1255`, colour *Warning*, label `Heavy`. The more severe band is first because the first match wins.
10. Leave the update rate alone: a linear gauge is a **Fast** component, about 250 ms, which suits a value read continuously.
11. Press **Preview**, choose a simulation scenario, and check that the gauge shows **Heavy** in the warning colour when the value goes above 1255. Leave preview.
12. Press **Validate**. The strip should say **READY TO PUBLISH**; fix any error it lists (a required slot left empty is the usual one).
13. Press **Save draft**, then **Publish…**. Enter the **Publication note** `Case weight gauge added for the giveaway check` and publish.
14. The dialog reports the new version (for example *Published as version 2*). Operators see the gauge the next time their client loads **Overview**, and the status bar reads `PACKAGING LINE › OVERVIEW v2`.

If the result is wrong, open **Publish…** again and use **Version history ▸ Roll back** on version 1.

Worked example: drawing the case magazine and checkweigher infeed

This example draws a small mimic on a new process page of the **Packaging line** project: the case erector's magazine as a vessel that fills, the checkweigher's drive as a symbol that shows whether it is running and turns while it is, the conveyor run between them with moving flow, a reading of the checkweigher's rate, and a hotspot that opens the checkweigher's detail card. It uses only signals the demonstration publishes:

Signal	Kind	Used for
`Packaging/Erector/MagazineLevel`	Number, %	The magazine's level and its low warning
`Packaging/Checkweigher/Running`	Boolean	The drive's colour

Signal	Kind	Used for
`Packaging/Checkweigher/Rate`	Number, cases/min	The drive's turning, the run's flow and the reading
`Packaging/Checkweigher/RatedSpeed`	Number, cases/min	The top of the rate range

Create the page.

1. In the project browser select **Packaging line** and press **New page…**.
2. Enter the **Name** `Infeed mimic`. The **Page id** becomes `infeed-mimic`.
3. Set **Type** to **Process**. **Layout** changes to **Process canvas**.
4. Under **Start from**, choose **Blank process page** and press **Create**. The editor opens on an empty canvas with the eight standard layers, and the palette opens its **Shapes and symbols**, **Process**, **Gauges and indicators** and **Navigation** groups.

Draw the magazine as a vessel.

5. Press `R` and drag a tall rectangle at the left of the canvas.
6. Open the **Signals** tab, type `MagazineLevel` in the search box, and drag `Packaging/Erector/MagazineLevel` onto the rectangle. **Driven by (1)** now holds a **Fill level** row.
7. Press its pencil to open it. Because the signal is in percent, the suggested range is right: **From value** `0`, **To value** `100`, **Gives** `0` **through** `1`, and **Hold at the ends when the value goes outside the range** ticked. The sentence under it reads *0 → 0, 100 → 1*.
8. To warn when the magazine runs low, set **What the new animation drives** to **Fill colour** and press **Animate**. In its **Driven by** source list choose **Signal**, then pick `Packaging/Erector/MagazineLevel` in **Signal**. Change the starter band to **Comparison** `<`, **Value** `25`, **Colour** *Warning*, label `Magazine low`. This is the band the demonstration's 3D cell uses on the same magazine.

Draw the drive as a symbol.

9. Press `S`. In the list beside the tool strip, choose *Geared electric motor*, then drag out the symbol to the right of the magazine.
10. Drag `Packaging/Checkweigher/Running` onto the symbol. Because the signal is a boolean, the Studio adds **Fill colour** with the bands *On* (Running colour) and *Off* (Unavailable colour).
11. Set **What the new animation drives** to **Keep turning** and press **Animate**. In its **Driven by** source list choose **Signal**, then pick `Packaging/Checkweigher/Rate`. Set **From value** `0` and **To value** to the checkweigher's rated speed (the **Signals** tab shows the current value of `Packaging/Checkweigher/RatedSpeed` beside it). Leave **Gives** `0` **through** `90` rpm.

Draw the conveyor run.

12. Press `P` and drag from the foot of the magazine to the foot of the symbol. The run starts as a default run inside that box.
13. Select the run. Drag its vertices so it leaves the magazine and arrives at the symbol; click a **+** (*Add a vertex here*) where it needs another corner, and double-click a vertex you do not want.
14. Set **What the new animation drives** to **Flow along the run** and press **Animate**. Choose **Signal** as its source and pick `Packaging/Checkweigher/Rate`, then set the same **From value** and **To value** as the drive. **Gives** `0` **through** `40` units per second is the default. When the rate is zero the flow stops.
15. With the run still selected, open the **Layers** tab and set **Layer** to **Process connections**, so the run sits behind the equipment.

Add the reading and the hotspot.

16. Press `T` and drag a label under the symbol. Drag `Packaging/Checkweigher/Rate` onto it: the Studio adds a **Text** animation, and the label shows the rate. In **Formatting**, set **Decimal places** to `1`.

17. Open the **Components** tab, type `hotspot`, and drag **Navigation hotspot** over the symbol. In the **Layers** tab set its **Layer** to **Navigation hotspots**.
18. In the hotspot's **Navigation** section set **Action** to **Show a page as a card**, **Target page** to `machine` and **Parameters** to `machine=Checkweigher`. Leave **Width** and **Height** blank: the card takes the Machine page's own canvas size. This is how the demonstration's **Line mimic** opens its machine cards.

Check and publish.

19. Press **Preview**. Choose a simulation scenario and watch the magazine level, the drive and the run respond to the simulated values. Leave preview.
20. Press **Validate**. Fix anything the strip lists at **Error**; the strip names the component each finding is about.
21. Press **Save draft**, then **Publish…**, with the **Publication note** `Infeed mimic`.
22. To put the page on the operators' rail, open **Navigation…** for the project, press **Add entry**, enter the **Label** `Infeed mimic`, choose the **Page** `infeed-mimic`, and press **Save navigation**.

Troubleshooting

Symptom	Cause and remedy
The Studio does not open	The account needs the edit pages permission. Permissions are the node's and take effect on the next request.
The page is read-only	Another author holds the lease. Wait, or use **Take over editing** with the permission.
Publish is refused	The page has validation errors; the **Validation** tab names what and where.
At Phone this component spans to column 12 in a 2-column grid, or *overlaps* another, at a breakpoint you never arranged	The component has no place of its own at that breakpoint and inherited its Full HD one. Press **Arrange missing resolutions** (on the page, or in the project browser for every page). Saving and publishing also fill it.
A component draws nothing	Its binding no longer resolves (a renamed tag is the usual cause), or a visibility rule is false.
The catalog is empty. Nothing can be picked until this deployment has one.	The node has no signals or assets yet; this is a commissioning step.
A value is stale	Check the source with **Connections** (**Admin ▸ I/O**, test the connection) before looking at the page.
The page is slow to open	Read **Page cost**: a large trend point budget, a wide semantic window, too many live components.
Components are missing from the canvas	Their layer is hidden with its eye in the **Layers** tab. Press the eye again to show it. Operators see the layer either way.
A click on a component selects something else, or nothing	The component is on a locked layer: the canvas passes presses through it. Unlock the layer in the **Layers** tab, or select the component in the tree.
A shape does not move or change colour	Open its **Driven by** section: an animation with no source, no bands or an empty input range does nothing, and the row says so. A value of bad quality also leaves a shape as drawn.
A card or dialog opens the wrong size	Set **Width** and **Height** in the component's **Navigation** section; blank takes the target page's canvas size.

Symptom	Cause and remedy
A pressed hotspot or button does nothing in **Preview**	Preview shows the page but does not act on presses. Publish, then try it on an operator screen.
Pressing a part of a machine does nothing	The part has no **Part detail** row, or the row names no signals.
Instances of a reusable component show *Missing composite*	Its definition is not on this installation, for example because the page was imported without it. Recreate it with the same id, or replace the instances. Deleting a composite from the **Library** does not do this.
Delete is unavailable in the **Library**	The entry is **Built in** or **Locked**, or you may not manage templates or reusable components. The menu says which.

Automation rules

> **Module: Web HMI.** This chapter covers the Automation workspace in the browser client, supplied by the Web HMI module. It is only present on installations licensed for that module.

A rule is what the plant does on its own when something happens: *when* case weight stays above 1255 g for ten seconds, *provided* the reading is below 1300 g (so it is a heavy case and not a sensor fault), *then* notify the shift. That is the whole model: **when** something happens, **provided** some conditions hold, **then** do these things.

Rules run on the node, not in your browser. Close every window and they keep going; open five and they do not run five times. Rules are part of the node's configuration, so they are activated as a numbered revision, compared, audited and rolled back like the alarm rules beside them.

Who may do what

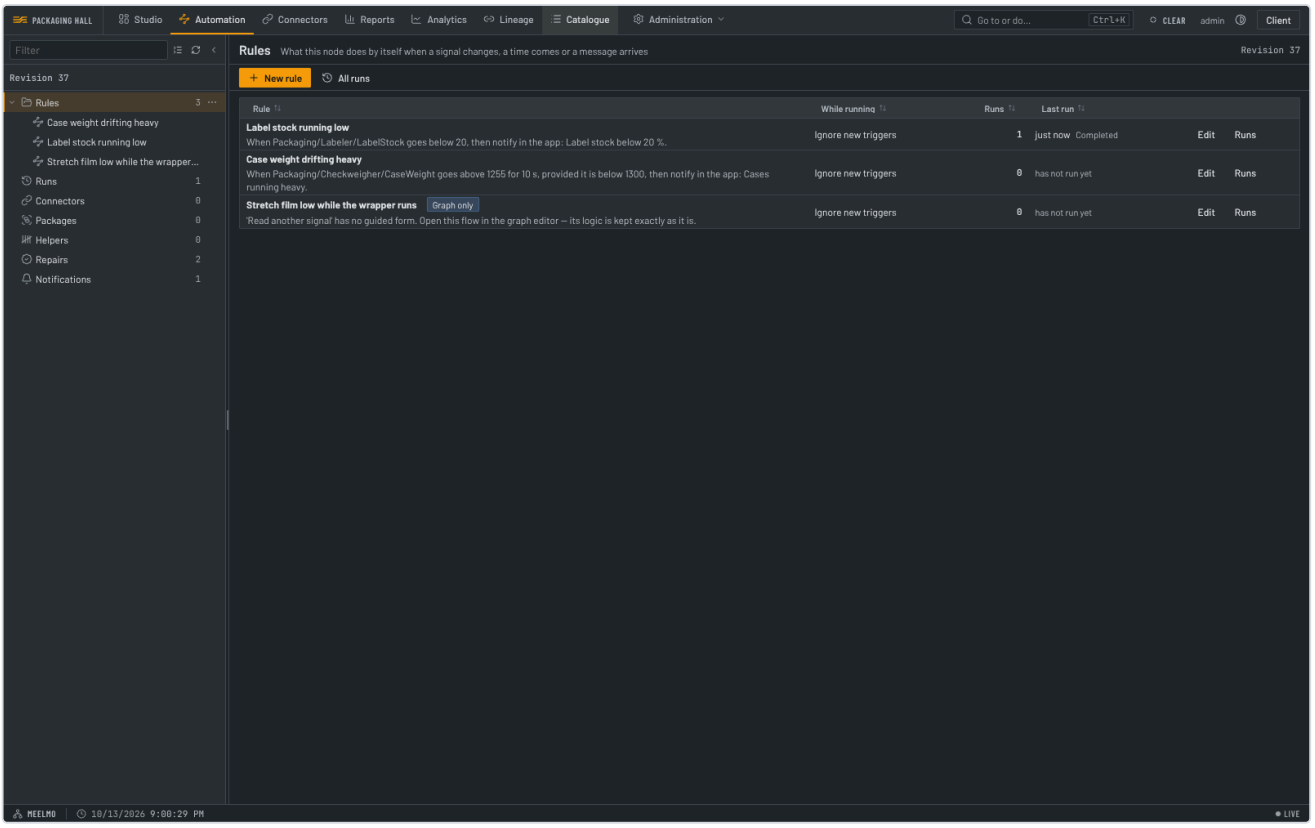
The **Automation** entry in the top navigation appears for anybody who may read rules. The buttons a person may not use are not drawn.

To	Permission	Seeded roles holding it
See rules, runs, helpers, repairs and notifications	Read automation (`Automation.Read`)	Operator, Engineer, Administrator
Draft and edit rules	Edit automation (`Automation.Edit`)	Engineer, Administrator
Activate, disable, roll back or delete rules	Activate automation (`Automation.Activate`)	Engineer, Administrator
Test a rule, or replay it against history	Test automation (`Automation.Test`)	Engineer, Administrator
Run a rule by hand against the plant, change a helper's value, import helper values	Run automation (`Automation.Run`)	Engineer, Administrator
Configure connectors	Configure connectors (`Connectors.Configure`)	Administrator
Create, upgrade and remove packages	Install packages (`Packages.Install`)	Administrator

An operator therefore sees what the rules did, which is how "why did the line stop?" gets answered on the shift it happened, and changes none of them. Commands a rule issues pass through the node's own authorization and validation, the same path a person's command takes, and are audited the same way.

If the workspace says *This host does not run automation.*, the web host is not attached to a node that runs rules; the message says which case applies.

The Automation workspace



The Automation workspace on the packaging line: the navigation tree on the left with the three demonstration rules, and the Rules list with each rule's sentence, execution policy, run count and last run

The workspace has a navigation tree on the left and a working area on the right.

The tree (with a **Filter** box, a list-view toggle, a refresh button and a collapse button at its top) shows the **Revision** the rules were activated as (`Revision 37`) and these entries, each with a count:

Entry	What it opens
Rules	The rule list; the rules themselves are listed under it. Each rule's ⋯ menu (or right-click) offers **Delete rule…**.
Runs	Every run of every rule.
Connectors	Where rules may reach outside the node, and what may reach in.
Packages	Ready-made rule sets, answered for this plant.
Helpers	The site's own counters, flags, meters and thresholds.
Repairs	What the node has noticed about its automation, and a setup checklist.
Notifications	What rules wanted somebody to know.

The Rules list (*What this node does by itself when a signal changes, a time comes or a message arrives*) has **New rule** and **All runs** at the top and one row per rule:

Column	Content
Rule	The rule's name and its sentence, for example *When Packaging/Labeler/LabelStock goes below 20, then notify in the app: Label stock below 20 %.* A rule that can only be drawn as a graph carries **Graph only** and says which block needs it.
While running	The execution policy (see *Execution policy*), here **Ignore new triggers**.

Column	Content
Runs	How many runs are recorded.
Last run	When, and how it ended (**2 min ago Completed**), or *has not run yet*.
Edit, **Runs**	Open the rule in the editor, or its runs.

The rule editor

Press **Edit** on a rule, or **New rule**, to open the editor. It has three columns and a findings bar.

Along the top: the rule's name, **Guided** and **Graph** (the view switch), **Undo** and **Redo** (each tooltip names what it will undo or redo), the **On** check box (whether the rule is switched on), **Test** and **Activate**.

The left column has two tabs. **Blocks** is the block catalogue, grouped as **When**, **Provided**, **Then**, **Shaping the value** and **Controlling the run**, with a **Find a block** box. A block this node cannot perform is shown greyed with **Not on this node** (in the figures, **Request a report** on a node that serves no reports); a block that calls a system outside the node carries **Reaches outside**. **Steps** lists the rule's own steps.

The right column has three tabs: **Properties** (the selected block's settings, or **Rule settings** when nothing is selected), **Test run** and **Against history**. Where the Studio Assistant is licensed and switched on, a fourth tab, **Assistant**, appears for authors. It proposes changes to the draft for you to review; it can never activate a rule.

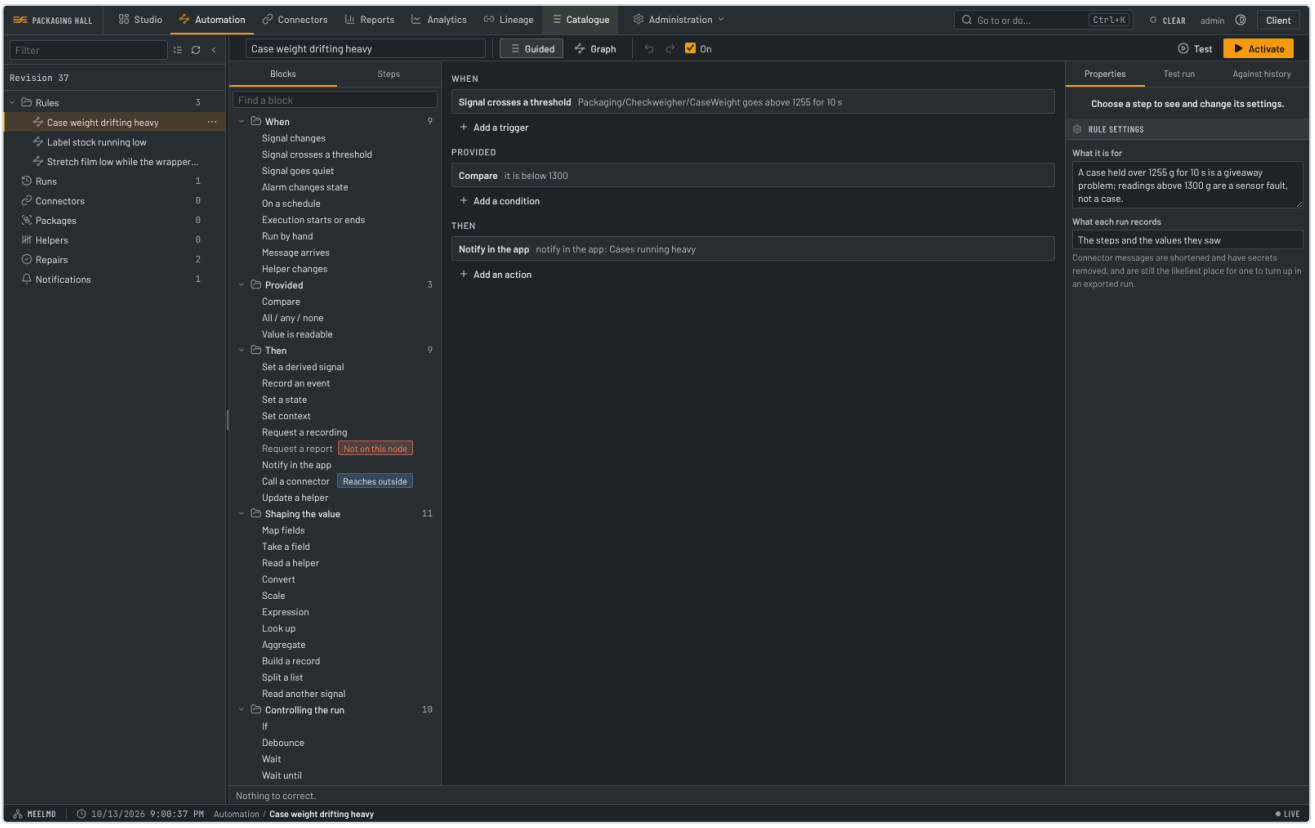
The findings bar along the bottom lists what is wrong with the rule. A finding tagged **Stops activation** must be fixed before **Activate** will put the rule into service; one tagged **Worth knowing** is advice. With nothing to report it says *Nothing to correct.*

Rule settings

With no block selected, **Properties** shows **Rule settings**:

Field	What it is
What it is for	The rule's description, shown to whoever reads it later.
What each run records	**Which steps ran, and how long they took** (no values); **The steps and the values they saw** (the default); or **Also what connectors sent and received** (shortened, with secrets removed). The third is never the default: a connector's message is the likeliest place for a secret to turn up in an exported run.

The guided editor



The guided editor on Case weight drifting heavy: When a signal crosses a threshold, Provided a comparison, Then notify in the app, with the block catalogue on the left and Rule settings on the right

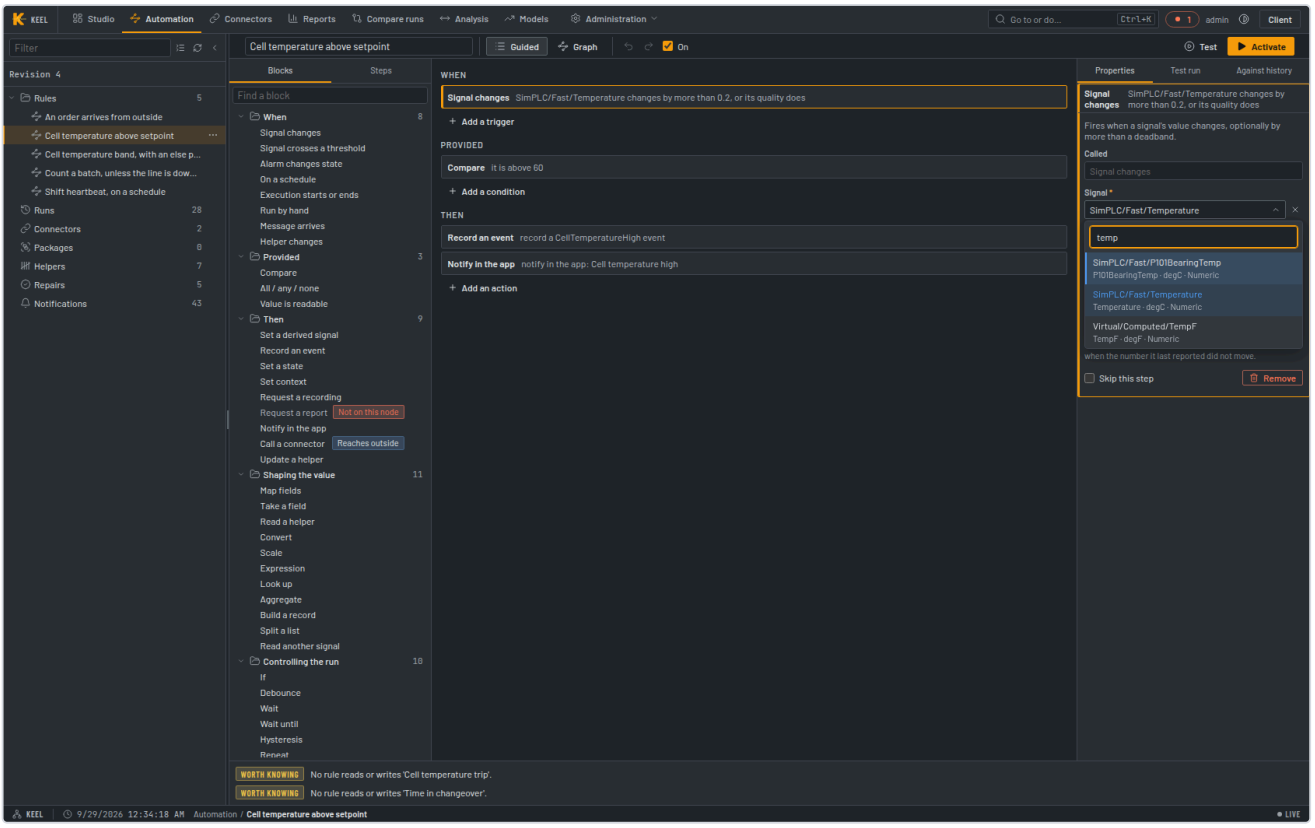
Guided reads the rule as a sentence in three sections:

- **When**: what starts it. **Add a trigger** offers the trigger blocks.
- **Provided**: what has to hold for it to go on. **Add a condition** offers the condition blocks. With none, *Nothing is checked, so this rule runs whenever it is triggered.*
- **Then**: what it does. **Add an action** offers the action blocks.

Each step reads as a phrase, for example **Signal crosses a threshold** *Packaging/Checkweigher/CaseWeight goes above 1255 for 10 s*. A step added here is wired into the rule for you in the right place: a condition added after the actions still goes before them. Click a step to open its settings under **Properties** on the right, where **Called** names the step, **Skip this step** leaves it out without deleting it, and **Remove** deletes it.

A new rule opens in the guided view. A rule needs at least one **When** step or it can never run (*Nothing starts this rule yet, so it will never run.*); validation reports it before you can activate.

Settings are chosen, not typed



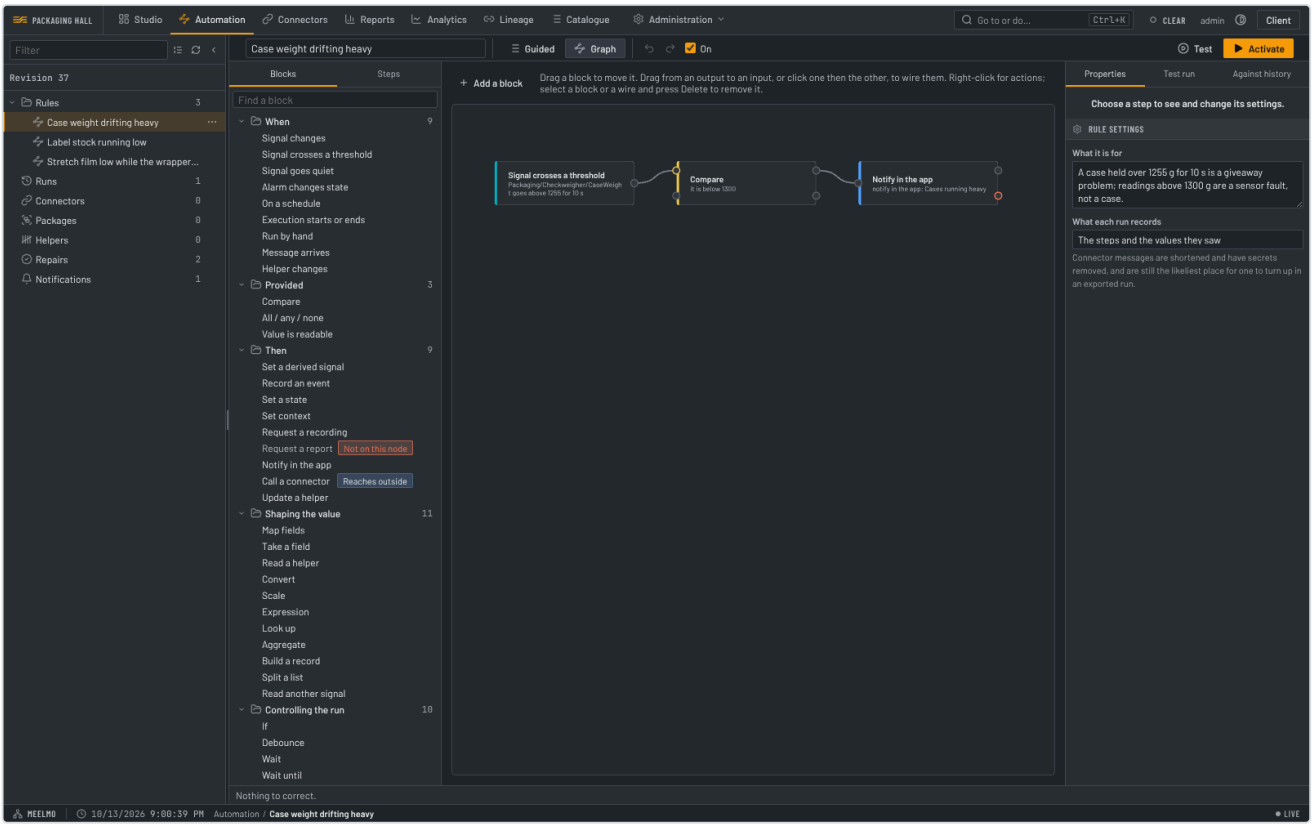
Choosing a signal: typing temp in the Signal list finds three signals, each with its unit and type

Every block's form is drawn from the block's own description, so any configurable setting has a field, and supported configuration never requires editing JSON. A setting that names something the node already has is picked from a list:

- **Signals** open a searchable list: type any part of the name (`weight` finds `Packaging/Checkweigher/CaseWeight`), 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*.
- **A schedule's days** are seven tick boxes; none ticked means every day.
- **A blank choice says what the node does with blank.** An alarm trigger with no transition chosen fires when the alarm is raised, and its form and sentence both say so.
- A value the node does not have, such as a signal since renamed or a connector since deleted, is kept and marked rather than cleared, so opening a rule never changes it.

The figure above was captured on the reference installation, whose signals are named `SimPLC/...`; on the packaging demonstration the same list offers the `Packaging/...` signals.

The graph editor



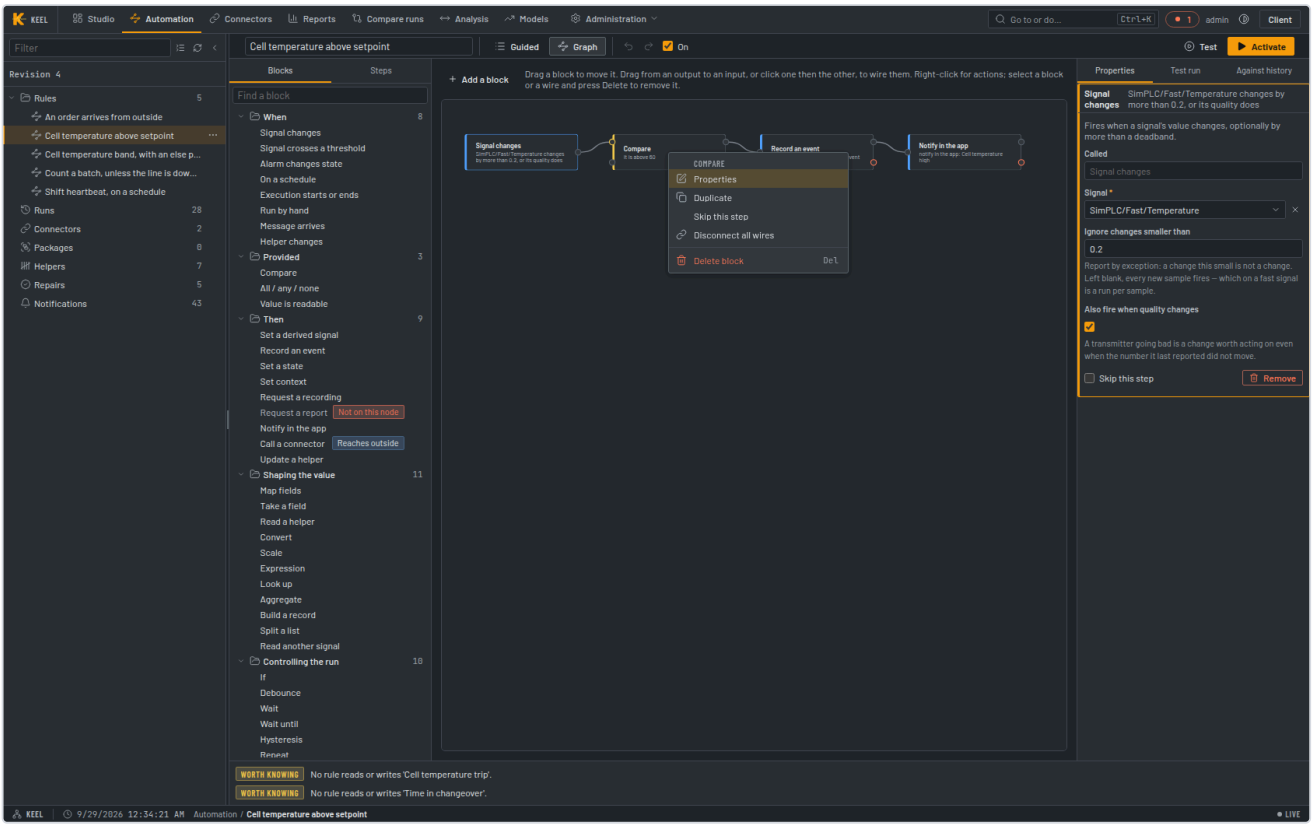
The graph editor on Case weight drifting heavy: the threshold trigger wired to the Compare block, wired to Notify in the app

Graph draws the same rule as blocks with typed ports and wires. It is one rule, not two: edit in either view and the other follows. The hint above the canvas summarises the gestures: *Drag a block to move it. Drag from an output to an input, or click one then the other, to wire them. Right-click for actions; select a block or a wire and press Delete to remove it.*

To	Do this
Add a block	**Add a block** above the canvas; or click a block in the **Blocks** list, or drag it onto the canvas where you want it; or right-click empty canvas ▸ **Add a block here**.
Move a block	Drag it; or select it and use the arrow keys (Shift for a longer step, Home for the left edge).
Wire two blocks	Drag from an output port (right-hand dot) to an input port (left-hand dot), or click one then the other. From the keyboard: Tab to an output port, press Enter or Space, Tab to an input port, press Enter or Space. Escape abandons a wire in flight.
Select a wire	Click it (it is highlighted), or Tab to it.
Delete a block or wire	Select it and press **Delete** or **Backspace**.
Let go of the selection	**Escape**.
Open the context menu	Right-click, or **Shift+F10**, or the Menu key, on whatever has the focus.

Ports are typed: the editor will not let you wire a text output into a number input without saying so, and validation catches anything the editor did not.

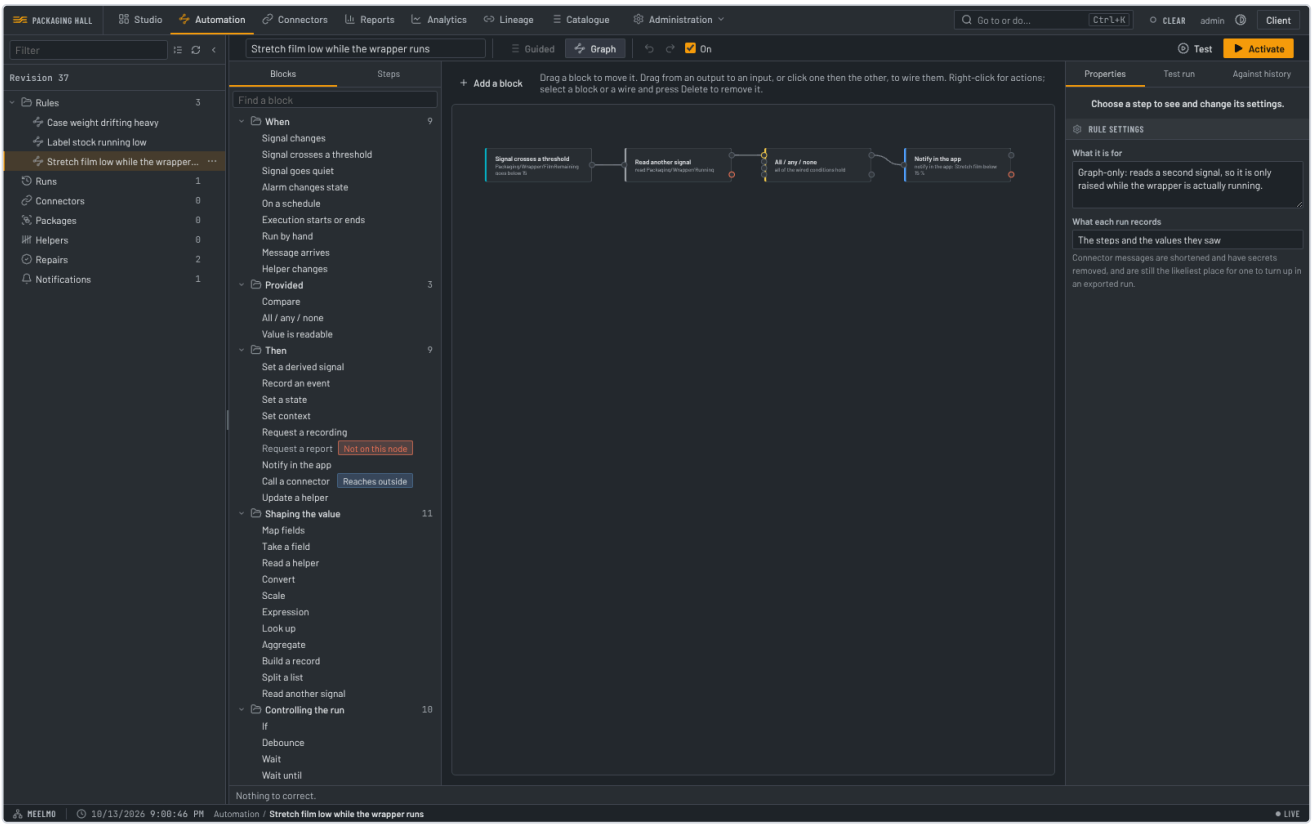
The context menu



The graph editor's context menu on a Compare block: Properties, Duplicate, Skip this step, Disconnect all wires and Delete block

Right-click on	Offers
A block	**Properties**, **Duplicate**, **Skip this step**, **Disconnect all wires**, **Delete block** (Del)
A wire	**Select the block it comes from**, **Select the block it goes to**, **Delete wire**
Empty canvas	**Add a block here**, **Undo**, **Redo**

Rules only the graph can draw



The graph-only demonstration rule Stretch film low while the wrapper runs: threshold trigger, Read another signal, All / any / none, Notify in the app; the Guided button is greyed

A rule with branches, loops, fragment calls, several triggers feeding different chains, a step nothing is wired to, or a block with no guided form can only be drawn as a graph. The **Guided** button is then greyed, its tooltip names the block that needs the graph, and the rule list shows **Graph only**. The guided view never shows part of such a rule and drops the rest, since that would let somebody activate a rule they were never shown in full.

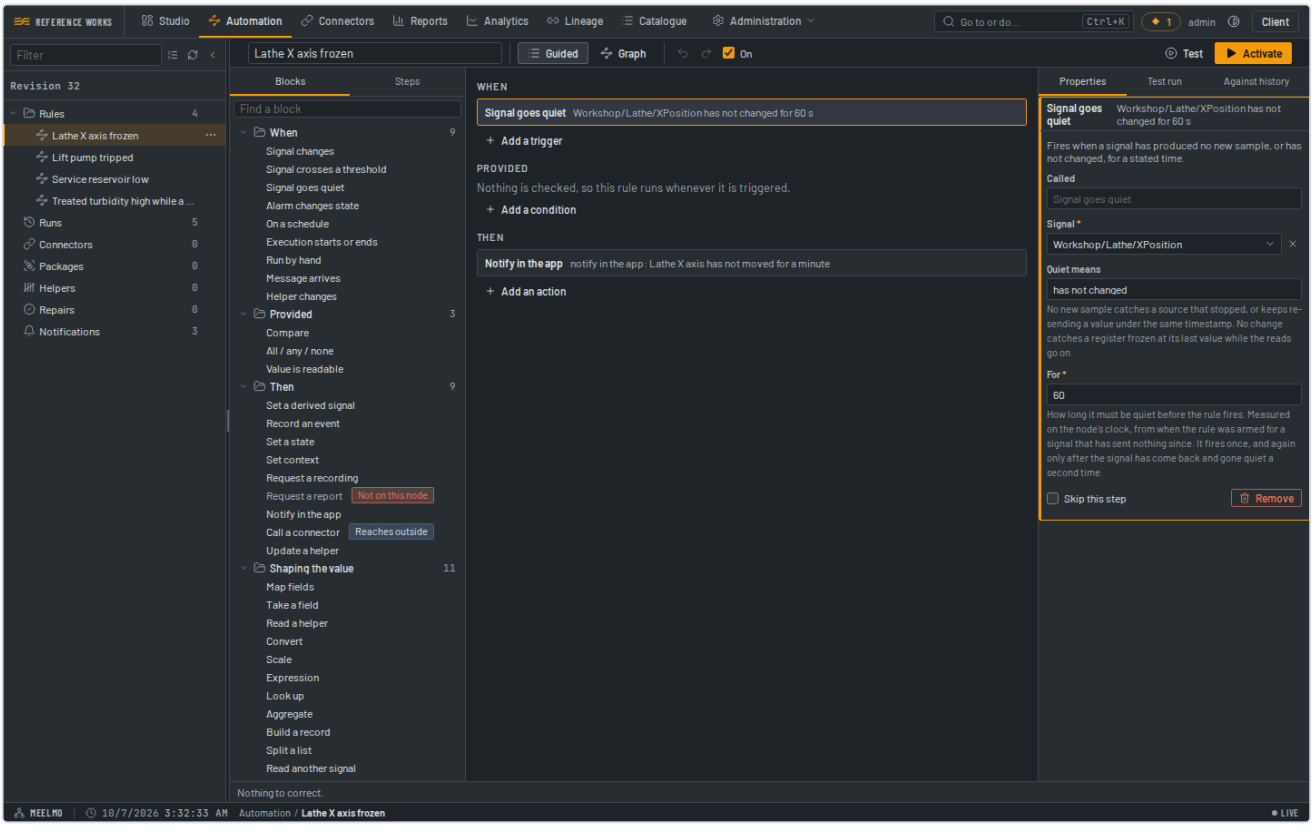
The block catalogue

The palette groups blocks as below. The id in the second column is how a run, a validation message and an exported rule name the block.

When: what starts a run

Block	Id	What it does
Signal changes	`trigger.signal`	Fires when a signal's value changes, optionally by more than a deadband (**Ignore changes smaller than**), and optionally when its quality changes (**Also fire when quality changes**).
Signal crosses a threshold	`trigger.threshold`	Fires on the edge where a signal crosses a value, with a direction, a hold time and hysteresis.
Signal goes quiet	`trigger.silence`	Fires when a signal has produced no new sample (**Quiet means** *has had no new sample*), or has not changed (*has not changed*), for **For** seconds, measured on the node's clock. It fires once, and again only after the signal has come back and gone quiet a second time.

Block	Id	What it does
Alarm changes state	`trigger.alarm`	Fires on an alarm's transition: raised, cleared, acknowledged or shelved.
On a schedule	`trigger.schedule`	Fires at a time of day, on chosen days, in the node's configured time zone.
Execution starts or ends	`trigger.execution`	Fires when the node's detection rules open or close an execution (an order, a batch).
Run by hand	`trigger.manual`	Fires when somebody runs the rule by hand, or when a test starts it.
Message arrives	`trigger.message`	Fires when a connector delivers a message: a webhook, an MQTT topic, a file.
Helper changes	`trigger.helper`	Fires when a counter, flag, selector, threshold or shift changes value.



The rule Lathe X axis frozen in the guided editor: When Signal goes quiet — Workshop/Lathe/XPosition has not changed for 60 s — Then Notify in the app, with the trigger's settings Signal, Quiet means and For in the property rail

Provided: whether it goes on

Block	Id	What it does
Compare	`condition.compare`	Compares a value against another value or a constant. You say whether an unreadable value counts as true or false; nothing is assumed.
All / any / none	`condition.boolean`	Combines up to four conditions.
Value is readable	`condition.quality`	True when a value is present and its quality is good, or good-or-uncertain.

Then: what the rule does

Block	Id	What it does
Set a derived signal	`action.signal`	Publishes a value as a signal the rest of the product can plot, record and alarm on.
Record an event	`action.event`	Writes an event into the node's history.
Set a state	`action.state`	Opens or closes a state interval on an asset.
Set context	`action.context`	Attaches a context value to an asset or an execution.
Request a recording	`action.record`	Asks the recorder to capture, including its pre-trigger buffer.
Request a report	`action.report`	Runs one of the node's report definitions and keeps the result.
Notify in the app	`action.notify`	Raises a notification in the browser and the Desktop.
Call a connector	`action.connector`	Sends a request through a connector and continues with the answer.
Update a helper	`action.helper`	Increments a counter, starts a timer, or sets a flag.

Shaping the value

Block	Id	What it does
Map fields	`transform.map`	Builds a record from another, field by field.
Take a field	`transform.field`	Reads one field out of a structured value, by path.
Read a helper	`transform.helper`	Reads a helper's current value.
Convert	`transform.convert`	Changes a value's type or unit, explicitly.
Scale	`transform.scale`	Maps a number from one range onto another, optionally clamped.
Expression	`transform.expression`	Evaluates a safe arithmetic expression over the wired inputs.
Look up	`transform.lookup`	Maps a value through a table you supply.
Aggregate	`transform.aggregate`	Reduces a list to one number: count, sum, mean, minimum, maximum.
Build a record	`transform.record`	Assembles a structured value from up to four inputs and constants.
Split a list	`transform.split`	Turns a list into one message per element, bounded.
Read another signal	`transform.signal`	Reads the last value the node saw for a signal the rule was not started by. Set **no older than** so an old reading gives *no value* rather than a stale number.

Controlling the run

Block	Id	What it does
If	`control.branch`	Sends the run one way or the other.
Debounce	`control.debounce`	Passes a value on only once it has stopped changing for a while.
Wait	`control.delay`	Holds the run for a fixed time, or limits how often it may pass.

Block	Id	What it does
Wait until	`control.waitFor`	Holds the run until a signal reaches a value, or gives up. Its *gave up* port is usually the interesting one.
Hysteresis	`control.hysteresis`	Turns a noisy number into a stable on/off with separate on and off points.
Repeat	`control.repeat`	Runs what follows a fixed number of times, or once per list element.
Call a subflow	`control.subflow`	Runs a reusable fragment and continues with what it returned.
Inspect	`control.inspect`	Records a value in the run and passes it on unchanged.
Subflow input / Subflow output	`control.input` / `control.output`	Where a fragment receives its value and what it returns.

Loops. Split a list and **Repeat** do not loop on their own: wire the last block of the loop body back to the driver's own input (`List` for a split, `Over` for a repeat). The returning wire asks for the next pass. Without it a split takes only the first element and a repeat never reaches its `When finished` port. Every run is also bounded at 500 steps.

Values carry quality. Every value carries its source time, its processing time, its quality and where it came from. A reading that cannot be believed does not become a zero: a transmitter failing low reads zero with bad quality, and a block refuses it rather than comparing it, and the run says it was the quality.

Execution policy

What happens when a rule is triggered again while a run is still in progress. The Rules list shows it in **While running**.

Policy	What it does	Use it when
Ignore new triggers	The new trigger is turned away and recorded as turned away.	One occurrence is one occurrence, such as a threshold that bounces during a hold time. All three demonstration rules use this.
Restart	The run in progress is cancelled and a new one starts.	A newer value should supersede the one being waited on.
Queue	Runs wait their turn, up to a bound; beyond it work is refused and the refusal recorded.	Order matters and the rate is survivable.
Run alongside	A bounded number of runs at once.	Runs are independent.

A run also has a **deadline**: a run that passes it is stopped, and the run says so. The default is five minutes; a rule that waits longer than that needs a longer deadline.

Testing a rule

Two ways to try a rule without touching the plant, both needing the test permission.

Test and Test run

Press **Test**. The rule as it stands on screen, including edits not yet activated, runs in an isolated runtime with its own state and its own clock, started as if it had been triggered. **Every external effect is suppressed**: no event is written, no notification sent, no connector called. Waits cost no real time, so a five-minute settle returns immediately. A connector that declares a sample answer answers with it, so a rule that calls out can be tested with nothing on the other end.

The **Test run** tab then shows the **Test result**:

- the **Simulated** chip and *Nothing here happened. Every action was suppressed at the node, so no event was written, no notification sent and no outside system called.*;
- a paragraph explaining the run;
- **It would have**: each action the rule would have taken, with its kind and summary, or *This run took no actions.*;
- **Steps**: the run step by step, as in *Runs and traces*.

A test starts the rule with nothing from a real trigger, so a rule whose condition needs a real reading may stop at that condition, and the result says why. To see what the rule does with real values, use **Against history**.

Against history

Against history asks the question a commissioning engineer has: would this rule have fired, and when? (*Runs this rule against what was actually recorded. Nothing it would do is done — the result says what it would have done, and when.*)

1. Open the **Against history** tab.
2. Set **From** and **To**, or press **Last hour**, **Last eight hours** or **Last day**.
3. Press **Replay it**. The node reads back the raw recorded samples for the signals the rule watches (*Watching: …*) and runs the rule through them.
4. Read the result: *It would have run {n} times.* or *It would not have run at all over that window.*, then a table of each run with **When**, **What happened** and the **Outcome**, each labelled **Simulated**.

Each run lands at the historical instant it would have happened, and waits take their time from the gaps between recorded samples. Refused rather than answered badly: a rule no signal starts (a schedule, a message: test it instead), a window wider than 31 days, and a read past 50,000 samples (*History held more than this replay read…*, narrow the window). One replay runs at a time on a node; acquisition comes first.

Worked example. Open **Case weight drifting heavy**, choose **Against history**, press **Last eight hours** and **Replay it**. If the heading reads *It would have run 3 times*, the table lists the three instants case weight held above 1255 g for 10 s while below 1300 g. If it would have run forty times, the threshold is too tight for this product and should be raised before the rule is left running.

Activating, revisions and deleting

Activate validates the rule and, if nothing stops it, puts it into service as a new revision of the node's configuration (*Activated as revision {n}.*). The button is disabled while a finding tagged **Stops activation** remains. The same revision mechanism covers the alarm rules beside it, so rules diff, roll back and audit the same way.

- The **On** check box decides whether the rule runs once activated. Unticking it and activating switches the rule off without deleting it.

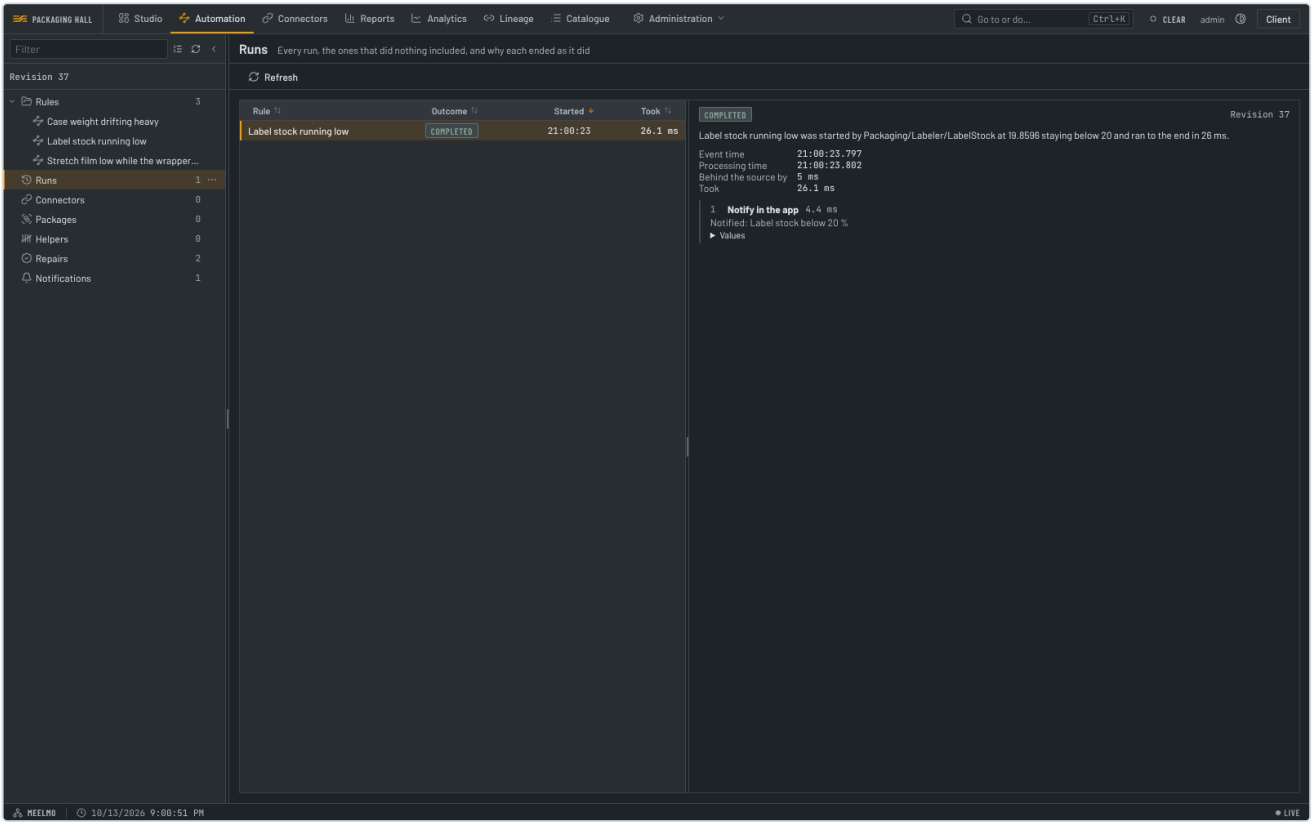
- A run is pinned to the revision it started under: activating mid-run does not change what a run in progress is doing.
- **Undo** and **Redo** cover every edit until activation. Activation clears the undo history, because what is running is now the document.

Deleting a rule:

1. In the tree, press the rule's **⋯** (or right-click it) and choose **Delete rule…**.
2. Confirm *Delete "{name}"?*: *The node stops running it as soon as it is deleted. The deletion is saved as a new revision, so it can be rolled back like any other change; the rule's past runs are kept.*
3. The result reads "*{name}" was deleted, as revision {n}.*

Deleting needs the activate permission. A rule a package created is removed with its package instance, under **Packages**.

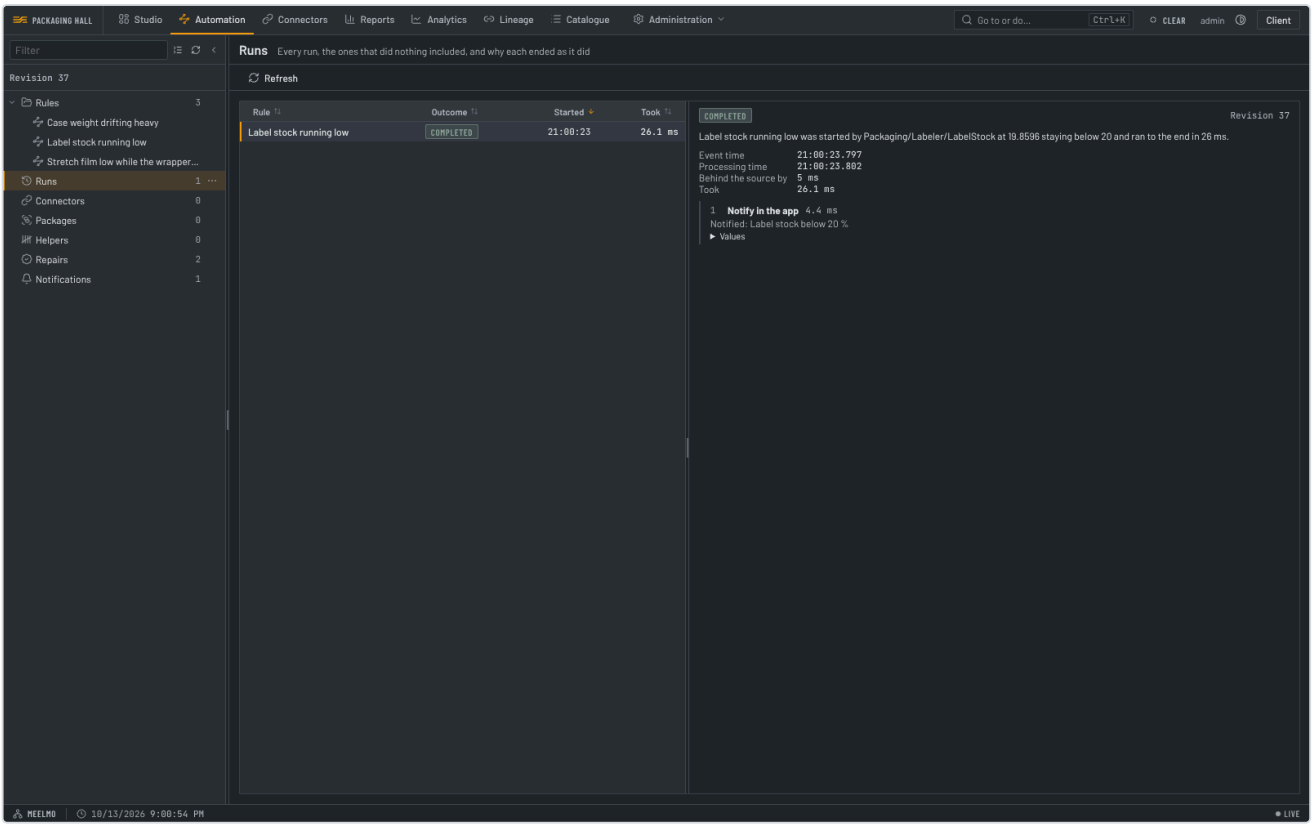
Runs and traces



The Runs page: one completed run of Case weight drifting heavy, and its explanation, times and steps on the right

Runs (*Every run, the ones that did nothing included, and why each ended as it did*) lists what the node did, newest first, with **Rule**, **Outcome**, **Started** and **Took**, and a **Refresh** button. Outcomes are **Completed**, **Condition not met**, **Not started** (turned away by the execution policy), **Past its deadline**, **Failed** and **Cancelled**.

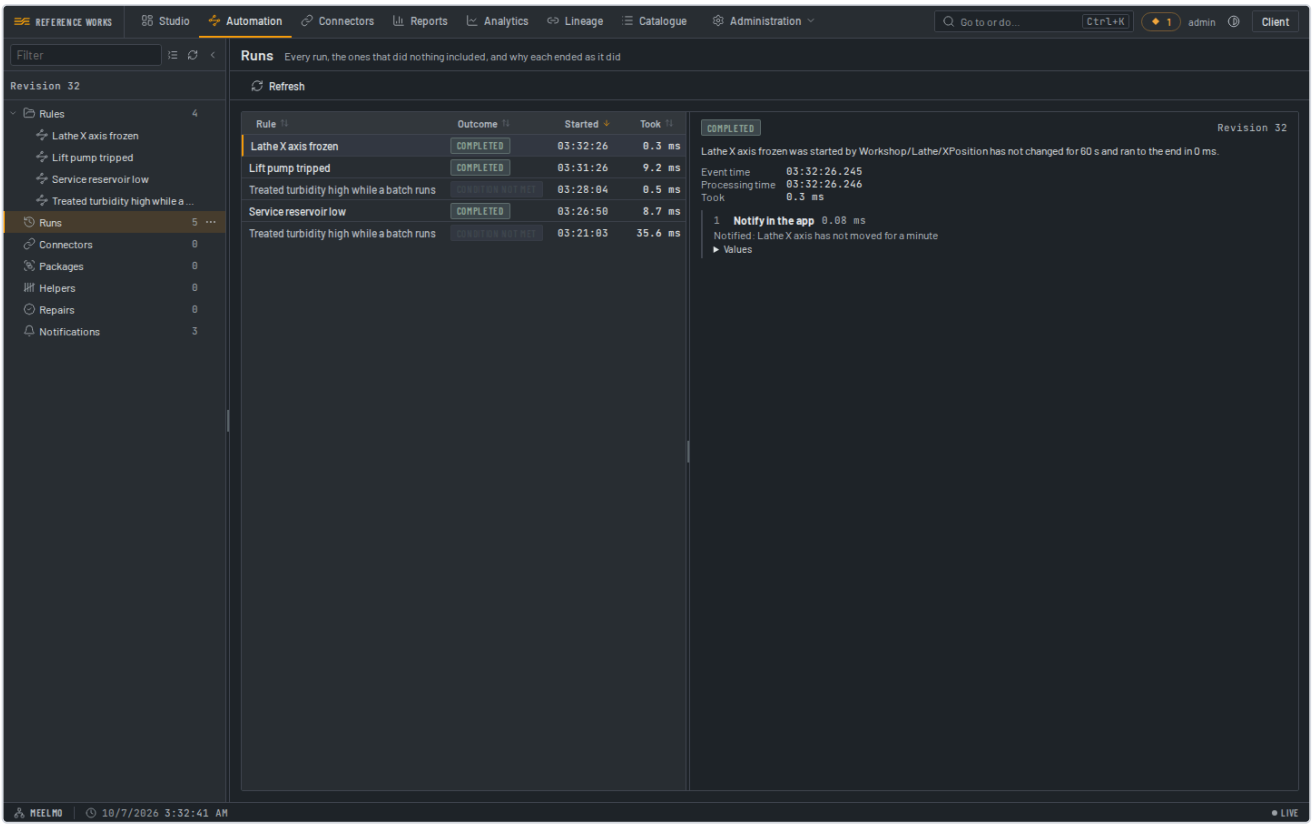
Runs that did nothing are recorded too, with their reason: "why did nothing happen?" is the question this page exists to answer. Each rule keeps its last 100 runs; the events, alarms and recordings rules produced are in the historian and are not bounded.



A run opened: Case weight drifting heavy started by CaseWeight at 1257.41 staying above 1255 for 10 s, with its event time, processing time, lag, duration, and the Compare and Notify in the app steps

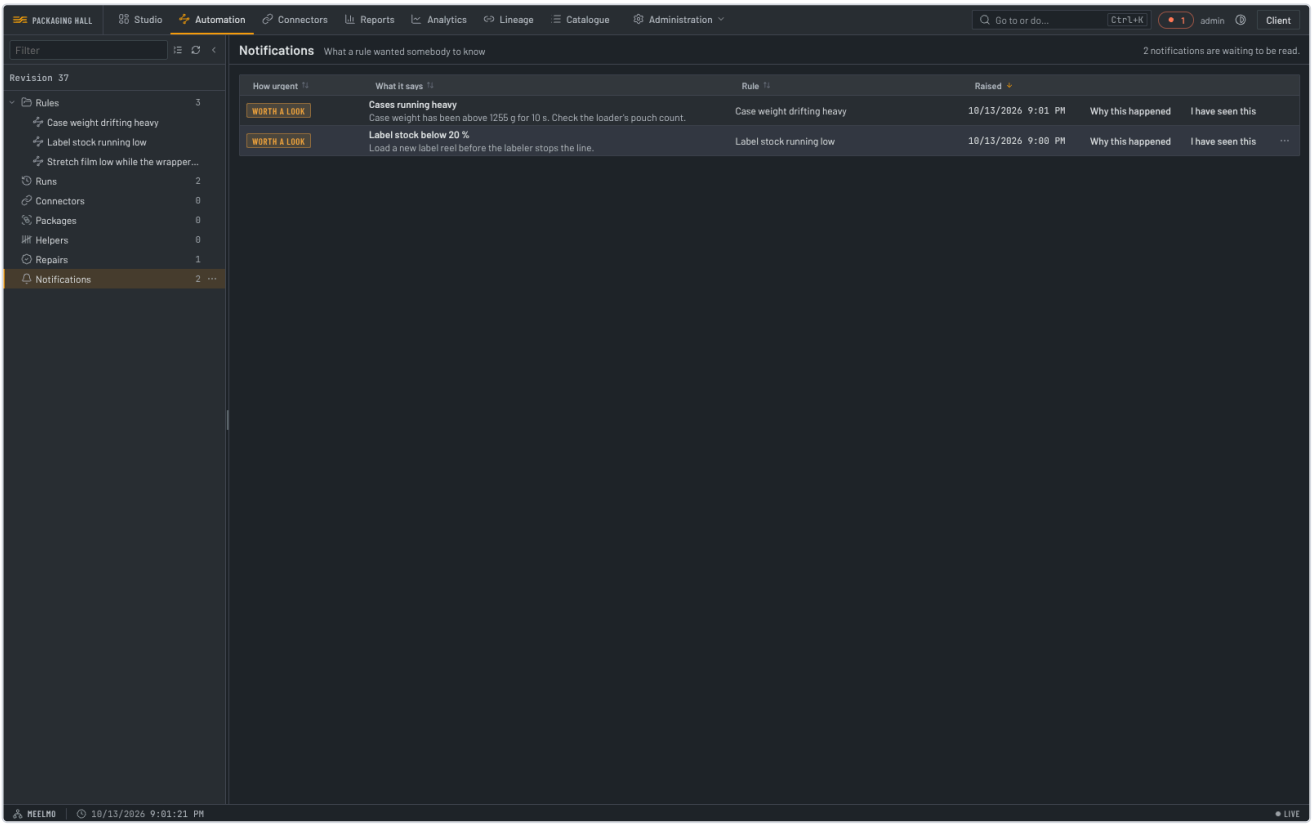
Select a run to read **What happened**:

- The outcome and the configuration **Revision** the run was pinned to.
- A sentence: *Case weight drifting heavy was started by Packaging/Checkweigher/CaseWeight at 1257.41 staying above 1255 for 10 s and ran to the end in 60 ms.*
- **Event time** (when the cause was true at its source), **Processing time** (when the node began the run), **Behind the source by** (the lag, `9 ms`) and **Took** (`60.4 ms`).
- Each step in order with its duration: **1 Compare** `4.04 ms`, *1257.41 is < 1300*; **2 Notify in the app** `10.52 ms`, *Notified: Cases running heavy*. **Values** under each step unfolds the inputs and outputs it saw (subject to **What each run records**).



The run of Lathe X axis frozen: completed, started by Workshop/Lathe/XPosition having not changed for 60 s, its Notify in the app step saying Lathe X axis has not moved for a minute; the works' other rules' runs are listed beside it

Notifications



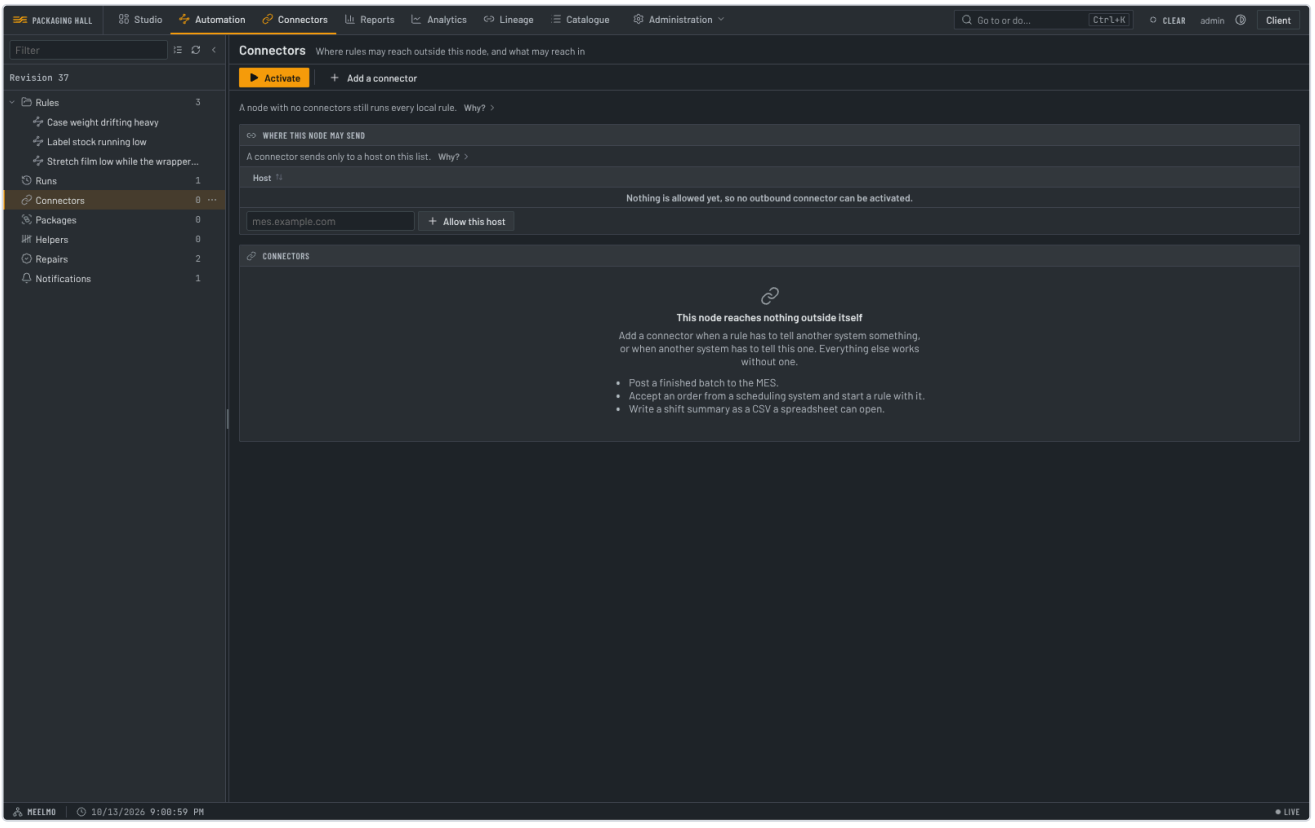
Notifications: one Worth a look notification, Cases running heavy, raised by Case weight drifting heavy, with Why this happened and I have seen this

Notifications (*What a rule wanted somebody to know*) lists what **Notify in the app** steps raised. The header counts what is waiting (*One notification is waiting to be read.*).

Column	Content
How urgent	**For information**, **Worth a look**, **Needs somebody**, **Needs somebody now**.
What it says	Title and body, for example **Cases running heavy** / *Case weight has been above 1255 g for 10 s. Check the loader's pouch count.*
Rule	The rule that raised it.
Raised	When.
Why this happened	Opens the run that decided it.
I have seen this	Takes it off the list.

A notification is not an alarm and not an audit record: dismissing it is not acknowledging anything. A notification raised by a test carries the **Simulated** chip. When nothing is waiting the page says *Nothing is waiting*.

Connectors



Connectors on the demonstration installation: Activate, Add a connector, the empty Where this node may send list, and This node reaches nothing outside itself

A connector is an optional way out to another system (an MES, a scheduling system, a file share) or a door in from one. *A node with no connectors still runs every local rule.* Losing a connector never stops local rules.

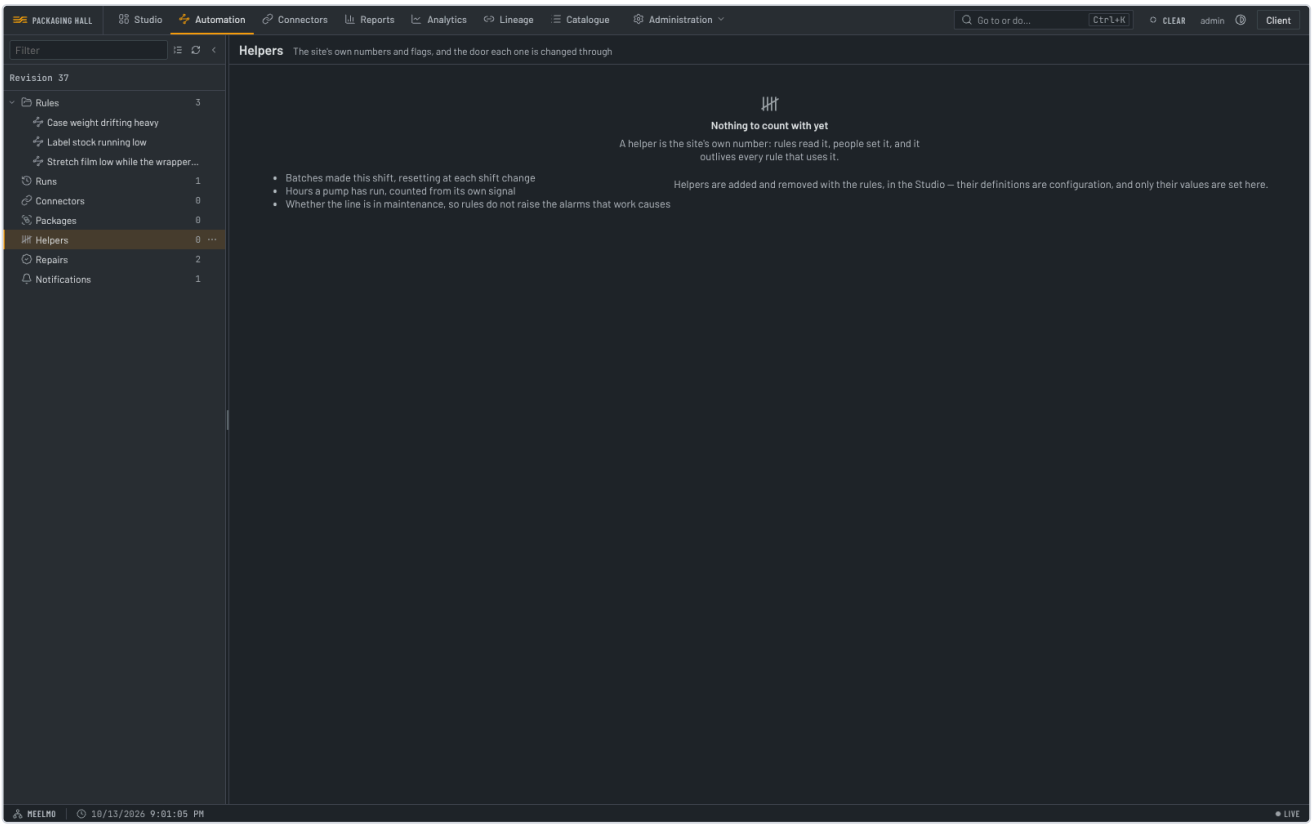
Where this node may send is the list of hosts any connector may send to (*A connector sends only to a host on this list.*). Type a host, for example `mes.example.com`, and press **Allow this host**; `*.example.com` covers one level of subdomain. With the list empty, *Nothing is allowed yet, so no outbound connector can be activated.*

Add a connector adds one to the draft. Its fields:

Field	Meaning
Called, **What it is for**	Name and description.
Kind	**Send a request**, **Receive a message**, **Read and write files**, or **Answer from this node** (no network; for building and demonstrating rules offline).
Address	For an outbound connector; its host must be on the list above.
Receives on	For an inbound connector: the path systems send to, with the shared secret in an `X-Meelmo-Token` header.
Folder, **Format**	For files: a folder inside the one the node allows, and the format named explicitly.
How it proves who it is	**Nothing**, **A user name and password**, **A bearer token**, **A key in a header** (**Header name**).
Credential	Sealed by the node and never shown again; leave empty to keep the stored one.
Operations a rule may name	One per line; empty allows any.
Gives up after (seconds)	The timeout.
Reads at most (bytes)	The size limit on an answer or an inbound message.
Messages a minute	For an inbound connector: the rate limit.
What it answers with when there is nothing to call	A JSON sample used by **Answer from this node** and by every test run.

Activate puts the connectors into service as a new revision (*Activated as revision {n}.*). A stored credential is never returned to a client and is not in the configuration, a diff, a backup or an export. The inbound door rate-limits, and a wrong path and a wrong secret give the same answer on purpose. Configuring connectors needs the connectors permission, held by the Administrator role.

Helpers



Helpers on the demonstration installation, with none defined yet: Nothing to count with yet, and the three examples

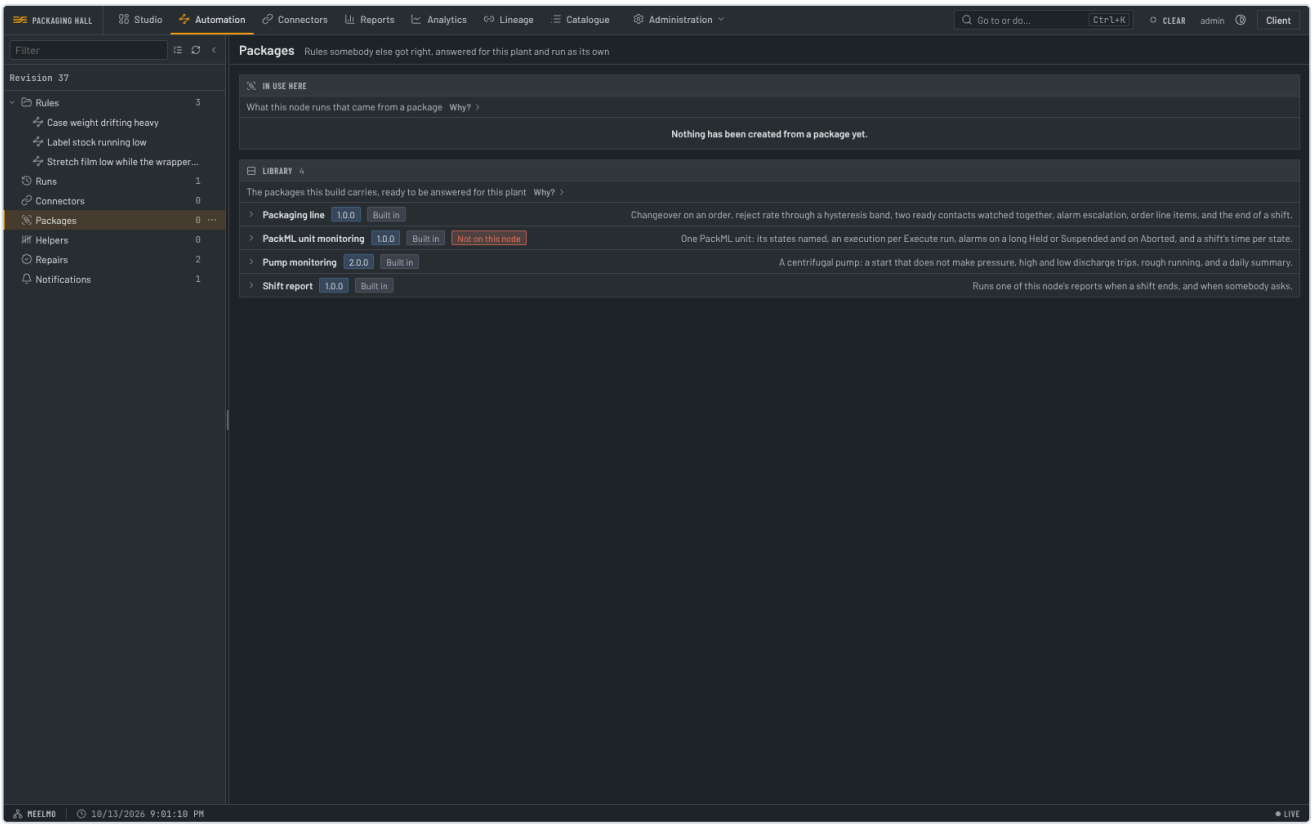
A helper is the site's own number: *rules read it, people set it, and it outlives every rule that uses it.* The kinds are **Counters**, **Timers**, **Operating-hour meters**, **Flags**, **Selectors**, **Thresholds** and **Shift patterns**. Examples: batches made this shift, resetting at each shift change; hours a pump has run, counted from its own signal; whether the line is in maintenance, so rules do not raise the alarms that work causes.

Helpers are added and removed with the rules, in the Studio — their definitions are configuration, and only their values are set here. For each helper the page shows its **Value**, **Last changed** and by whom, and **Read or written by** (the rules that use it), with the operations its kind allows: **Add one**, **Take one off**, **Back to the start**, **Set**, **Start**, **Stop**, **Turn on**, **Turn off**. Changing a helper's value needs the run permission.

- A **threshold** helper lets an engineer tune a trip point without editing a rule.
- An **operating-hour meter** is *The machine's own* record and refuses to be set by hand.
- A **flag** such as "maintenance mode" suppresses rules while people work, without disabling them.

Moving a node: a node backup does not contain helper values. Before moving to new hardware, use **Export the values** and import them on the other side, choosing *merge* (carry counts onto a node that has been running) or *replace* (restore).

Packages



Packages: nothing in use yet, and the library of three built-in packages: Packaging line 1.0.0, Pump monitoring 2.0.0 and Shift report 1.0.0

A package is a set of rules somebody already got right, with the questions a site has to answer (*Rules somebody else got right, answered for this plant and run as its own*). Three are **Built in**, compiled into the build, so they are available on a node that has never seen a network:

Package	Version	What it is
Packaging line	1.0.0	A whole line: changeover on an order, reject rate through a hysteresis band, two ready contacts watched together, alarm escalation, order line items, and the end of a shift.
Pump monitoring	2.0.0	One centrifugal pump: a start that does not make pressure, high and low discharge trips, rough running, and a daily summary. The one to read while learning the blocks.
Shift report	1.0.0	Runs one of the node's reports when a shift ends, and when somebody asks. The smallest complete package.

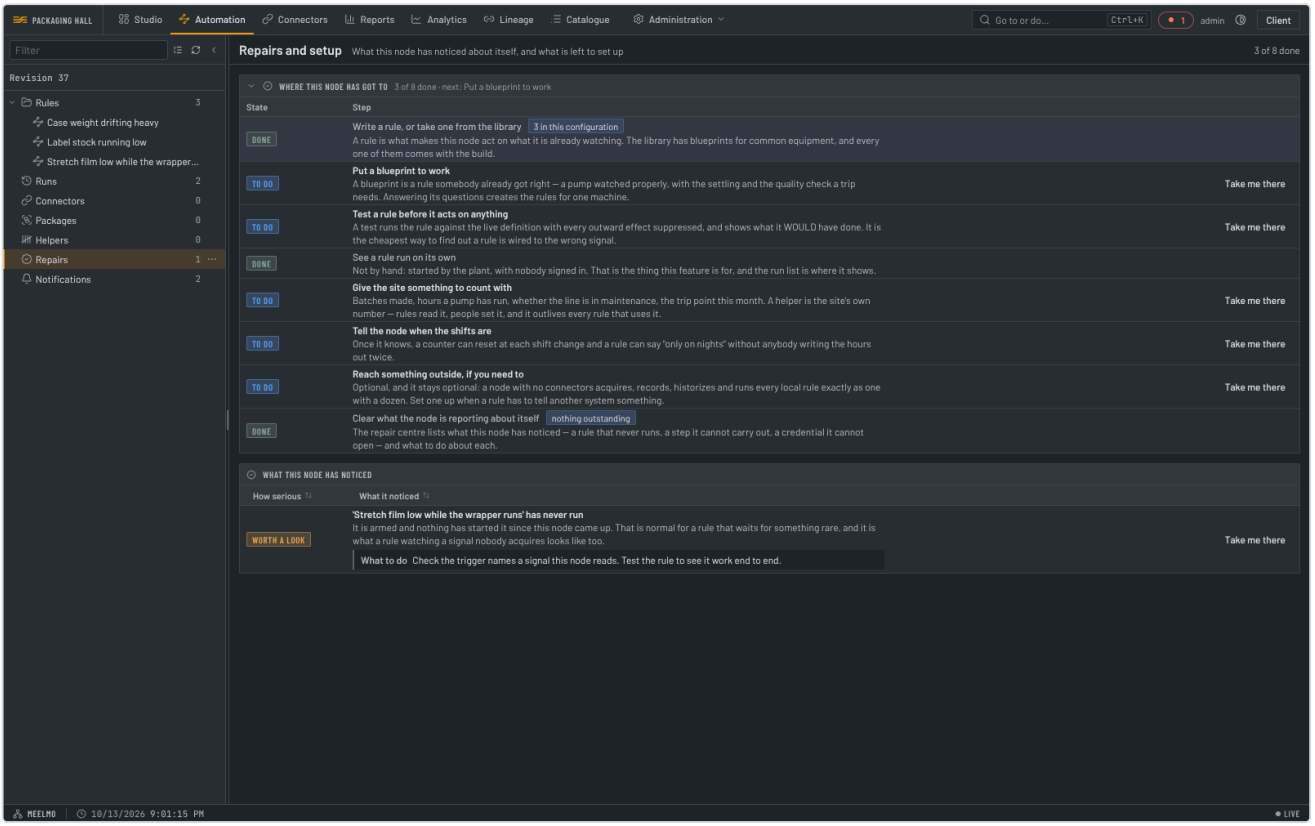
To use one:

1. Under **Library**, expand the package. It lists what it contains (*Rules: {n} · Reusable parts: {n}*) and the questions.
2. Enter **Call this one** (for example `Feed pump P-101`) and answer the questions. Most have a default; a question left blank switches off the rules that needed it, visibly, with the reason written on the rule.
3. Press **Check the answers** to see **This would create**.
4. Press **Create and activate**. The result reads *Created as revision {n}. Rules added: {n}*.

The instance then appears under **In use here**. It is pinned to the package version it came from; when a newer version is available, **See what an upgrade would change** shows the change before anything happens, and **Upgrade to {version}** keeps any edits you made to your own copy. **Remove** deletes the

instance and every rule it created, as a new revision. A package is refused whole if any rule in it needs something the node lacks (*Needs {x}, which this node has none of.*). Creating from a package needs the packages permission.

Repairs and setup



Repairs and setup: the eight-step checklist with three done, and two findings that the label stock and stretch film rules have never run

Repairs and setup (*What this node has noticed about itself, and what is left to set up*) has two parts. It is worked out fresh each time it opens: nothing is stored and nothing is acknowledged, so a finding goes away only when its cause does.

Where this node has got to is a checklist read off the node (**3 of 8 done**): write a rule or take one from the library; put a blueprint (package) to work; test a rule before it acts on anything; see a rule run on its own; give the site something to count with (helpers); tell the node when the shifts are; reach something outside, if you need to; clear what the node is reporting about itself. Each step is **Done** or **To do**, and **Take me there** opens the place to do it.

What this node has noticed lists findings worst first, with **How serious** (**Problem**, **Worth a look**, **Worth knowing**), **What it noticed**, **What to do** and **Take me there**. In the figure, '*Label stock running low*' *has never run* and '*Stretch film low while the wrapper runs*' *has never run*: both are armed and nothing has triggered them since the node came up, which is normal for a rule waiting for something rare, and is also what a rule watching a signal nobody acquires looks like. What to do: check the trigger names a signal this node reads, and test the rule.

The three demonstration rules

Loading the packaging demonstration (**Help ▸ Demonstration** in the Desktop, or Meelmo Setup's demonstration choice) puts three rules into service beside it, from `examples/demo/packaging.automation.json`. Their ids begin `demo-`, and loading another demonstration plant replaces them without touching any rule the site wrote. All three are switched on and use **Ignore new triggers**.

Label stock running low

`demo-packaging-label-stock`: *Warns the line before the labeler runs out, while there is still time to load a reel.*

Step	Block	Settings
When	Signal crosses a threshold	Signal `Packaging/Labeler/LabelStock`, threshold `20`, falling edge, hysteresis `5`
Then	Notify in the app	Title *Label stock below 20 %*, body *Load a new label reel before the labeler stops the line.*, severity warning

It fires once as label stock falls through 20 %, and cannot fire again until stock has risen back above 25 % (the hysteresis), so a reading hovering around 20 % does not notify repeatedly. It is deliberately earlier than the project's **Label stock low** alarm, which is raised below 15 %: the rule gives the operator time to act before the alarm.

Case weight drifting heavy

`demo-packaging-case-weight`: *A case held over 1255 g for 10 s is a giveaway problem; readings above 1300 g are a sensor fault, not a case.*

Step	Block	Settings
When	Signal crosses a threshold	Signal `Packaging/Checkweigher/CaseWeight`, threshold `1255`, rising edge, hysteresis `3`, held for `10` s
Provided	Compare	The value is below `1300`; an unreadable value counts as false
Then	Notify in the app	Title *Cases running heavy*, body *Case weight has been above 1255 g for 10 s. Check the loader's pouch count.*, severity warning

The hold time means a single heavy case does not fire it; weight has to stay above 1255 g for ten seconds. The **Provided** step separates a heavy case from a failed load cell: a reading above 1300 g, or one with bad quality, stops the run at the condition (**Condition not met**) instead of notifying. This is the rule shown in the Runs and Notifications figures: a run started at 1257.41 g, passed *1257.41 is < 1300*, and raised **Cases running heavy**.

Stretch film low while the wrapper runs

`demo-packaging-film-low`: *Graph-only: reads a second signal, so it is only raised while the wrapper is actually running.*

Step	Block	Settings
When	Signal crosses a threshold	Signal `Packaging/Wrapper/FilmRemaining`, threshold `15`, falling edge, hysteresis `3`

Step	Block	Settings
Shaping	Read another signal	Signal `Packaging/Wrapper/Running`, no older than `30` s
Provided	All / any / none	Mode **all** of the wired conditions
Then	Notify in the app	Title *Stretch film below 15 %*, body *Swap the film roll at the next pallet change.*, severity information

It is started by the film level but judged on whether the wrapper is running, which is two signals, and **Read another signal** has no guided form, so the rule is **Graph only**. A wrapper stopped for a changeover with a low roll does not notify; one running on a low roll does. If the running signal is more than 30 s old, it reads as no value rather than a stale "running", and the rule does not notify. The project's **Stretch film low** alarm, below 12 %, stays as the backstop.

Worked example: changing a demonstration rule

To make **Label stock running low** warn earlier, at 25 %:

1. Open **Automation ▸ Rules** and press **Edit** on **Label stock running low**.
2. In **Guided**, click the **When** step and set its threshold to `25` under **Properties**.
3. Click the **Then** step and change the title to `Label stock below 25 %`.
4. Open **Against history**, press **Last day**, then **Replay it**, and check how many times it would have run.
5. Check the findings bar reads *Nothing to correct.*, then press **Activate**. The result reads *Activated as revision {n}.*

The change is a new configuration revision: it appears in the node's revision history and can be rolled back. Loading the demonstration again replaces it with the shipped version.

Troubleshooting

Symptom	Cause and remedy
This rule has never run. (in Repairs)	Its trigger names a signal the node does not acquire, or it is waiting for something rare. Check the signal; test the rule.
A rule fires far more often than expected	Its trigger has no deadband or hysteresis, so it fires on every sample. Set one, or settle the reading with **Debounce**.
A rule fires once and then never again	Under **Ignore new triggers**, a run stuck waiting turns everything away behind it. Look for **Not started** runs, and check the deadline.
A value reads as absent where you expected a number	By design: a signal the node has not seen, or not recently enough for **no older than**, is absent rather than zero. The run says which.
Activate is disabled	A finding tagged **Stops activation** remains; read the findings bar.
Guided is greyed	The rule has something only the graph can draw; the tooltip names it.

Reports and analysis in the browser

> **Module: Web HMI.** This chapter covers the **Reports**, **Analytics**, **Catalogue** and **Lineage** areas of the browser client. It is present only on installations licensed for the Web HMI module. The same reports, datasets and analyses are available from the Desktop in every installation; the base chapters *Reports*, *Datasets* and *Analysis* describe them there. The **Models** and **Edge models** tabs that appear beside the analysis pages belong to the Models module and is described with that module.

A site whose people work from a browser — a shift supervisor in the plant office, a quality engineer on a laptop, a manager on a tablet — needs the plant's documents and its "why did that happen" questions without an engineering workstation. The browser client gives them four places in the top bar:

Top-bar entry	Address	What it is for
Reports	`/reports`	Browsing the reports this installation holds, running one over a window and taking it away as PDF or Excel, and composing reports in the **Report Studio**
Analytics	`/analytics`	The analysis tools in one place: whether the process can hold its spec (**Process capability**, `/analytics/process`), ranking what changed around a moment (**What changed**, `/investigate`), finding which signals have anything to do with each other (**What relates**, `/investigate/relations`), and comparing a run — an order, a batch, a lot, whatever this plant's rules detect — against reference runs that define what good looks like (**Compare runs**, `/analysis/runs`)
Catalogue	`/catalogue`	Finding a signal by the plant's word for it, and keeping the glossary and the signals' owners — see The signal catalogue
Lineage	`/lineage`	Where an item comes from and everything that depends on it, and what a change to it would reach — see Lineage

Somebody whose work is analysis rather than operating the plant can make **Analytics** their start page, so signing in opens it instead of the operator screens — see *Analytics as your start page* below.

Every example in this chapter uses the demonstration packaging line: connection `Packaging`, areas such as `Erector`, `BottomSealer`, `Checkweigher` and `Palletiser`, and signals such as `Packaging/Checkweigher/CaseWeight`, `Packaging/Checkweigher/Rate` and `Packaging/Checkweigher/Running`. The demonstration installs nine datasets (`packaging-line-shift`, `packaging-line-runs`, `packaging-line-events` and others) and two reports: **Packaging line (simulated) — shift report** and **Packaging line (simulated) — run report**. The analytics figures also read the packaging hall's production database: its MES records a works order per row — the SKU, the line (Line 1 is the demonstration line, Lines 2 and 3 its neighbours), the cases made and rejected and the QA lab's average case weight — and its checkweigher keeps a batch log of its own. Two run sources on **Administration ▸ Run sources** read them into the datasets `orders` and `checkweigher-batches`, the case weight has a customer spec per SKU, and a second node, the reference works (a water treatment works), is linked under the prefix `Works`.

Who sees what

The browser offers only what the signed-in person may use. Every control is also checked on the server; hiding a button is courtesy, not enforcement.

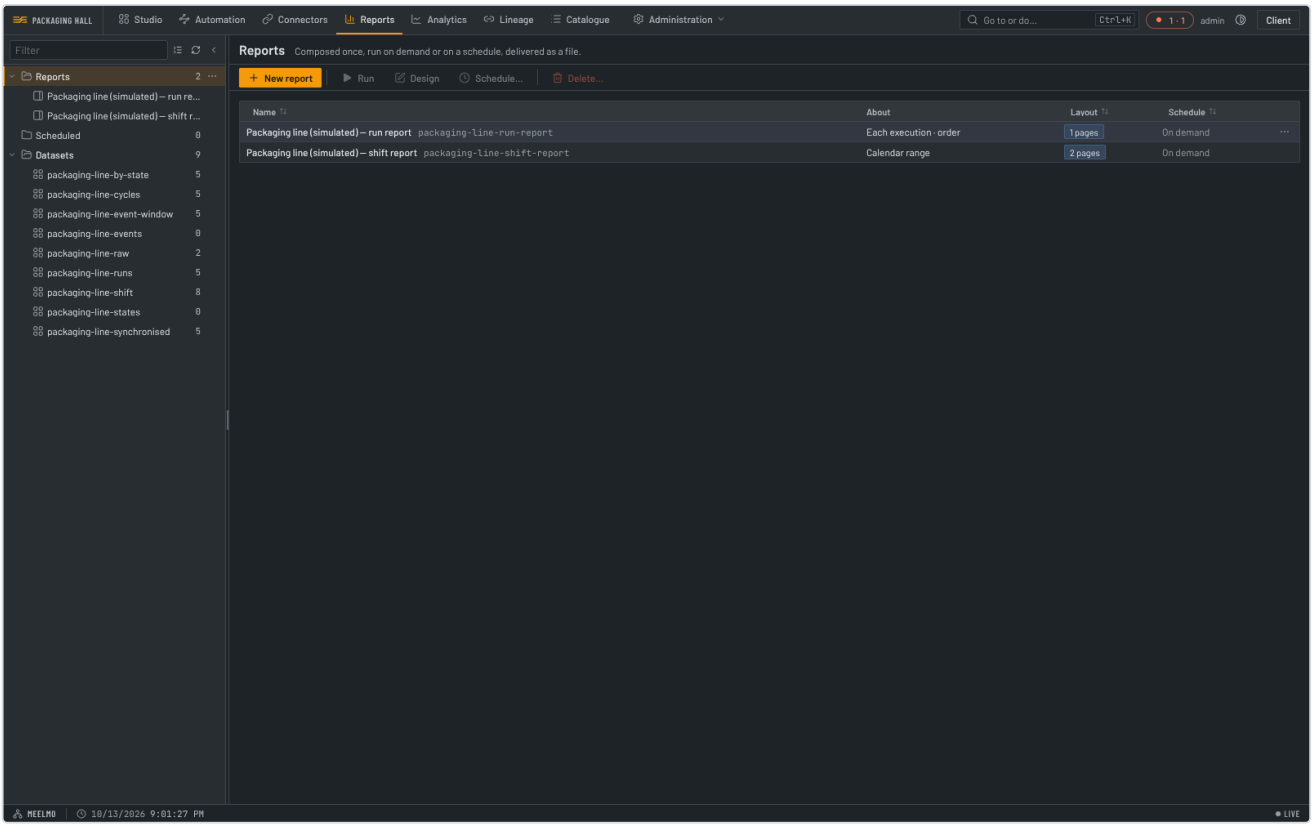
Area	Shown to	Checked on the server
Reports tab	anybody the node granted `Reports.View`	listing and opening reports
Run, **Run report**, **PDF**, **Excel**	`Reports.Run`	rendering a report (also needed for the designer's preview)
New report, **Design**	`Reports.Design`	creating and changing report definitions
Delete…	`Reports.Manage`	deleting a definition
Schedule… and the **Schedule** section of **Report settings**	`Reports.Design` and `Reports.Schedule`	changing when a report runs by itself and where it goes
Analytics (the hub)	everybody signed in	nothing: it only lists the tools below that this person may open
Process capability, **What changed**, **What relates** (and the **Models** tab)	accounts that may read history	every analysis route checks the history permission first; a study of a run summary also needs the events permission, since its items are the plant's runs
Compare runs	everybody signed in	reading the runs and their signals
Catalogue, **Lineage**	everybody signed in	each answers only with the signals and items this person may read; changing the glossary needs the permission to change configuration

An Engineer holds all five report permissions, an Operator holds **view** and **run**, a Viewer holds **view** only. These are a *person's* permissions granted at the node, not a service account's.

Analysis pages also ask the licence each time they open. Where the licence does not include historical queries, the page says so — "This installation's license does not include historical queries, and every analysis here is made from them." — with a link **See what the license covers**, instead of offering a form that could only be refused. Someone without the history permission is told "You are not permitted to read history…" and that an administrator can grant it.

The Reports area

Reports in the top bar opens a workbench: an explorer tree on the left, and on the right whatever is chosen in it.



The Reports area with the Reports folder chosen: the explorer tree on the left lists the two demonstration reports, the Scheduled folder and the nine saved datasets; the table on the right shows each report's name, what it is about, its layout and its schedule.

The explorer

The tree has three folders, each with its count on the right:

Folder	Holds	Right-hand figure on each row
Reports	every report the installation holds	— (the folder shows how many)
Scheduled	the reports whose schedule is switched on	the first time of day it runs, `HH:mm`
Datasets	the saved datasets reports can read	how many columns the dataset has

Above the tree:

Control	Does
Filter	Narrows the tree to rows whose name contains what you type
Collapse-all button	Closes every folder
Refresh button	Reads the reports and datasets again from the node
Fold button (`‹`)	Folds the explorer away to give the content the full width; on a phone the explorer becomes a sheet

Choosing a row writes it into the address (`/reports?open=…`), so a link to a particular report or dataset can be sent to somebody. Pressing **Enter** or double-clicking a report opens it where it is worked on: the Report Studio for someone who may design, the viewer for someone who may only run it. A right click (or the ⋯

on a row, or the Menu key) shows the same acts as the toolbar: **Run**, **Design**, **Schedule…** and **Delete…**; on the **Reports** and **Datasets** folders it offers **New report**.

The toolbar

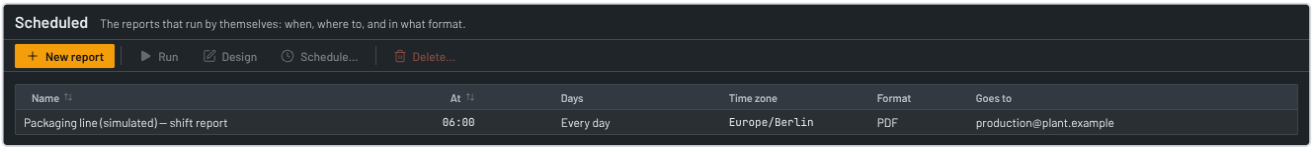
The toolbar under the page title always shows the same acts, in the order you use them:

Button	Does
New report	Opens the Report Studio on a new report
Run	Opens the chosen report in the viewer (greyed out, with "Choose a report first.", until a report is chosen)
Design	Opens the chosen report in the Report Studio
Schedule…	Opens the chosen report in the Report Studio with **Report settings** showing, at its schedule
Delete…	Asks "Delete “…”?" — any schedule that runs it stops, and a deleted report cannot be brought back — then deletes it

What each folder shows

Reports folder. A table of every report: **Name** (the title, with the report's identifier in a monospace face beside it, for example **Packaging line (simulated) — shift report** `packaging-line-shift-report`), **About** (what the report covers — *Calendar range*, *Each execution · order*, *Each state interval*, *Around each event*), **Layout** (`2 pages` for a report composed on pages, or `N sections` for one composed in the older section form, which is tinted and says it can be converted) and **Schedule** (*On demand* or *Scheduled*). Columns with arrows sort.

Scheduled folder. One row per report that runs by itself: **Name**, **At** (times of day), **Days**, **Time zone**, **Format** (*Web page*, *PDF* or *Excel*) and **Goes to** (the e-mail address, directory or webhook it is delivered to). When nothing is scheduled the table says "No report runs by itself yet."



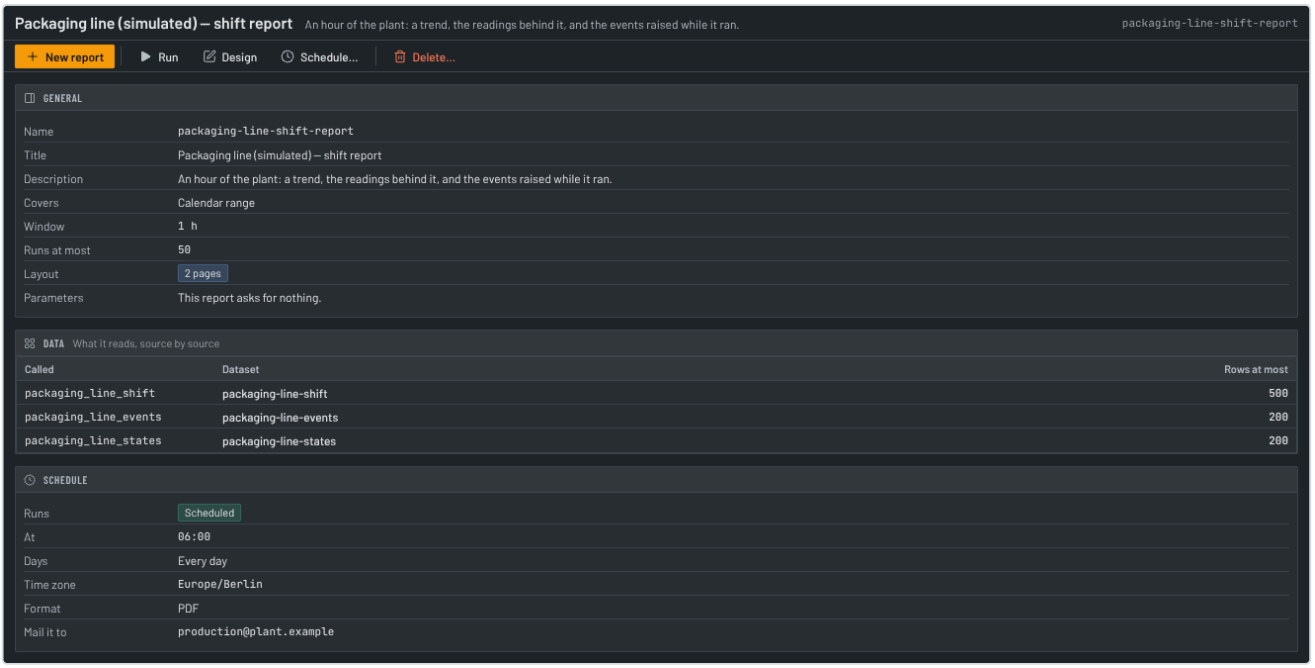
The Scheduled folder chosen: the shift report runs at 06:00 every day, Europe/Berlin, as a PDF mailed to production@plant.example.

Datasets folder. One row per saved dataset: **Name**, **Description**, **Sampling** (*As recorded*, *Every 1 min*, *Per acquisition batch*…), **Signals** (how many) and **Used by** (how many reports read it).

A report chosen. Three read-only panels:

- **General** — **Name**, **Title**, **Description**, **Covers** (the scope), **Of type** and **Where** when set, **Window** (the default look-back, for example `1 h` for the shift report and `24 h` for the run report), **Runs at most**, **Layout** and **Parameters** (the parameter names, or "This report asks for nothing.").
- **Data** — "What it reads, source by source": each data source's identifier (**Called**), the **Dataset** it names and **Rows at most**. A dataset the installation no longer holds is marked **Missing**. Choosing a row opens that dataset.
- **Schedule** — **Runs** (*Scheduled* or *On demand*) and, when scheduled, **At**, **Days**, **Time zone** and **Format**, followed by whichever of **Mail it to**, **Write it to** and **Post it to** are set.

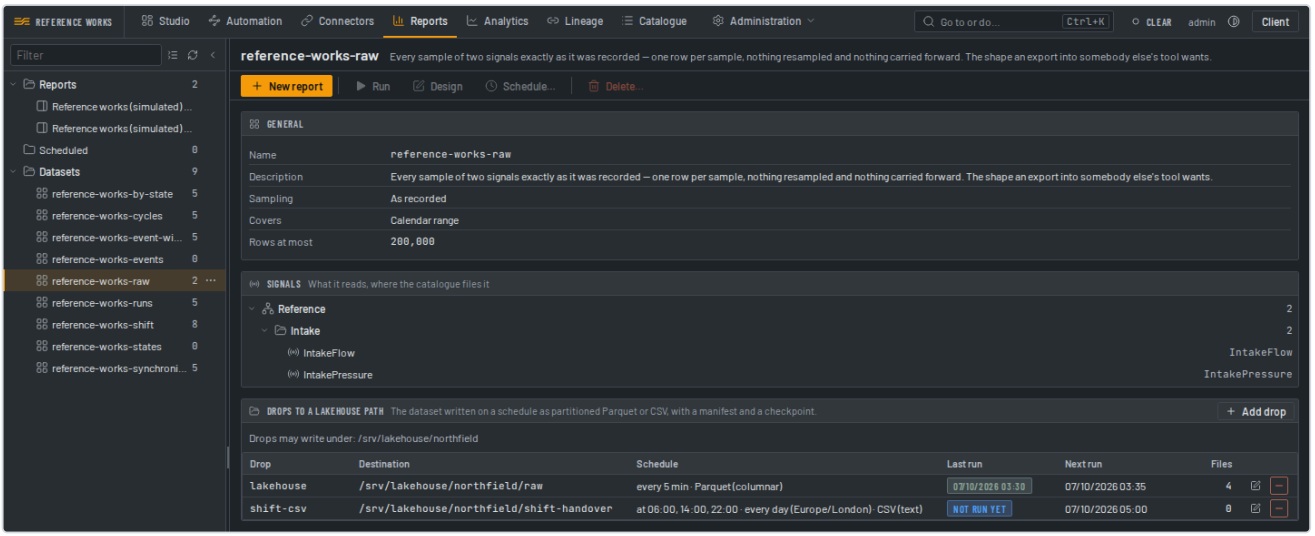
Nothing is edited in these panels. Every one of these facts is changed in the Report Studio, where the change is one undo step and is validated before it is saved.



The shift report chosen in the explorer: General with its name, title, description, Calendar range, a window of 1 h, at most 50 runs, 2 pages and no parameters; Data with its three sources and the datasets they name; Schedule with Scheduled, 06:00, every day, Europe/Berlin, PDF and Mail it to production@plant.example.

A dataset chosen. Its general facts (**Name**, **Description**, **Covers**, **Where**, **Sampling**, **Rows at most**), the **Signals** it reads grouped as the catalogue files them ("What it reads, where the catalogue files it") and **Used by** — the reports that read it, or "No report reads it yet.". Datasets are defined on the Desktop (**Data ▸ Datasets**); the browser shows them so that a report author can see what a report will read.

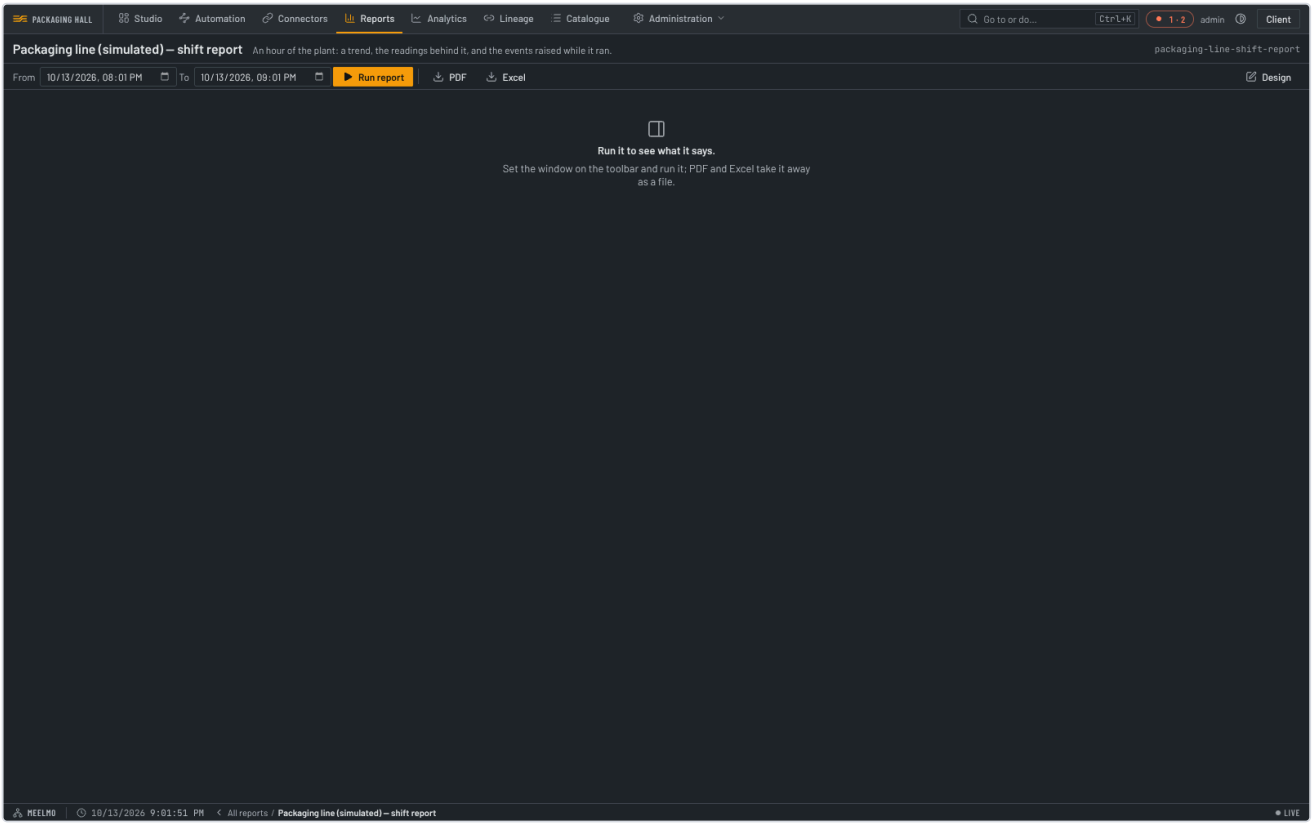
Below them, **Drops to a lakehouse path** — the one part of a dataset the browser edits. A drop writes the dataset on a schedule as partitioned Parquet or CSV, with a manifest and a checkpoint, for Databricks, Snowflake, Fabric or DuckDB to read: **Destination**, **File format**, **Partition by signal**, and **Runs** every so many minutes or at times of day in a time zone. The table says each drop's **Last run**, **Next run** and **Files**. A drop writes only beneath the directories the machine's administrator listed in `dataset-drop-roots.json` beside the node, which the panel names; with no list, no drop writes. A destination outside them is refused when the drop is saved, also on a plant server's web host, which asks the node. **Add drop**, change and remove need the permission to change the node's definitions — the node's I/O permission, which an engineer has — where the node keeps them, as it does on every host that serves one; elsewhere the permission to manage datasets.



The dataset reference-works-raw chosen: its general facts, the two signals it reads, and Drops to a lakehouse path — drops may write under /srv/lakehouse/northfield; lakehouse writes Parquet partitioned by signal every 5 minutes and has written six files, shift-csv writes CSV at 06:00, 14:00 and 22:00 Europe/London and has not run yet

Running a report and taking it away

Run (or a double click, for someone who may only run reports) opens the report viewer at `/reports/view/<name>`.



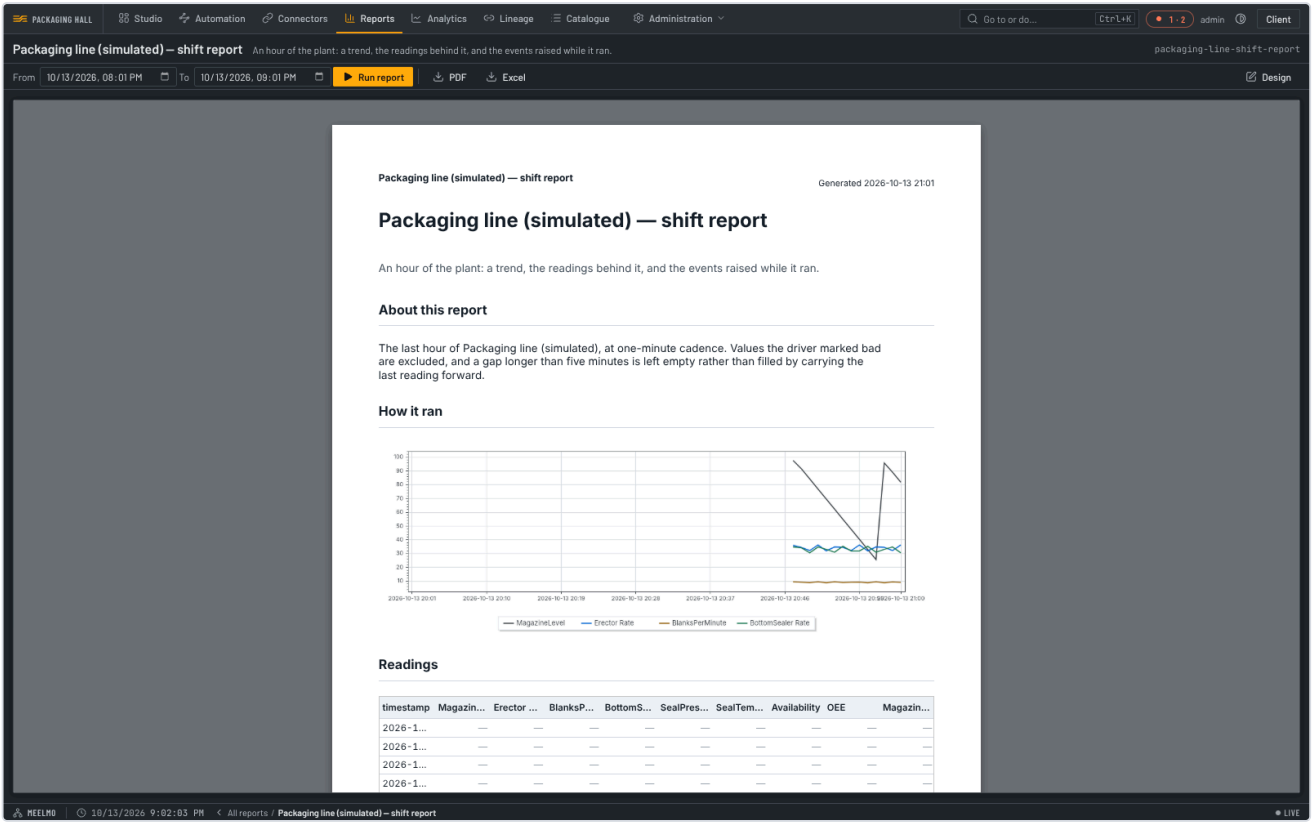
The shift report in the viewer before it has been run: the From and To boxes are set to the report's default window of one hour, with Run report, PDF and Excel beside them and Design on the right.

The page title is the report's title and description, with its identifier at the right. The breadcrumb in the status bar (**All reports / …**) goes back to the list.

Control	Does
From, **To**	The window the report covers. When the page opens, **To** is now and **From** is now less the report's own window (one hour for the shift report, 24 hours for the run report)
Run report	Sends the report to the server, which composes and renders it, and shows the result below. While it runs the button shows **Running…**
PDF	Renders the report over the same window and saves it as a PDF file
Excel	Renders it as an Excel workbook (`.xlsx`) and saves it
Design	Opens the report in the Report Studio (only for someone who may design)
Parameters panel	Appears under the toolbar only when the report asks something of whoever runs it. One row per parameter, in the reader's language, with the default already filled in. Choices appear as a list, numbers as a number box, dates and date-times as pickers, yes/no as a check box

Before anything is run the page says **Run it to see what it says.** — "Set the window on the toolbar and run it; PDF and Excel take it away as a file."

The report appears in a frame on a white ground, because it is paper. What you see is exactly what a schedule would deliver: the same composer and the same renderer produce the browser view, the PDF and the scheduled file. The files are named after the report and the end of the window, for example `packaging-line-shift-report-20260929-0700.pdf`.



The shift report after Run report: its header and generated stamp, the note about how the hour was read, the trend of the charted signals, the table of one-minute readings and the events raised during the hour.

Every report ends with a **coverage** note on its last page — for example "8 of 9 readings recorded; 1 not". It cannot be switched off: a report that silently averaged over the readings it had would be a report somebody signs without knowing what was missing.

A value with no reading prints the element's no-value text (an em dash by default), never zero.

Worked example: the morning shift report as a PDF

The supervisor of the packaging line wants the last hour of the early shift, 06:00 to 07:00, as a PDF for the shift log.

1. Click **Reports** in the top bar.
2. In the explorer, choose **Packaging line (simulated) — shift report**. The **General** panel shows **Covers** *Calendar range* and **Window** `1 h`; **Data** shows that it reads `packaging-line-shift`, `packaging-line-events` and `packaging-line-states`.
3. Click **Run** on the toolbar.
4. Set **From** to `29/09/2026 06:00` and **To** to `29/09/2026 07:00`.
5. Click **Run report**. The report appears: the note "The last hour of Packaging line (simulated), at one-minute cadence…", the **How it ran** trend of `MagazineLevel`, `Erector Rate`, `BlanksPerMinute` and `BottomSealer Rate`, the **Readings** table and **What was raised**.
6. Read the coverage note at the end. If it says readings were missing, check the historian before signing the report.
7. Click **PDF**. The browser saves `packaging-line-shift-report-20260929-0700.pdf` (the time in the name is the end of the window).
8. For the planner who wants the numbers in a spreadsheet, click **Excel** as well.

If the run is refused, a red banner **The report could not be run.** says why in the server's own words — typically a required parameter that was not supplied (the report names it), or a report the validator refuses (open it in the Report Studio and read **Problems**). A PDF refused with "empty boxes" contains characters the embedded font cannot draw; Excel and the browser view use the reader's own fonts.

The Report Studio

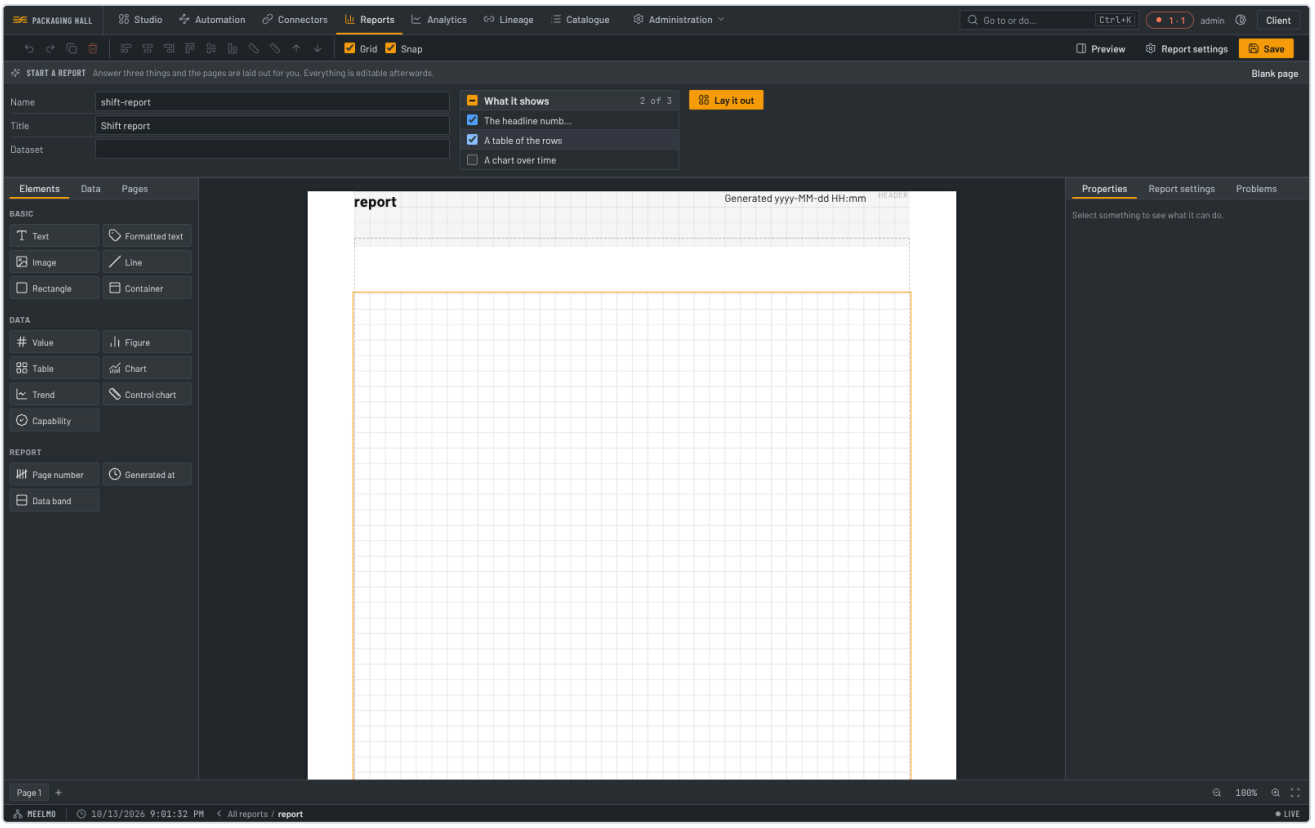
New report on the Reports toolbar, or **Design** on a report, opens the Report Studio at `/reports/design` (or `/reports/design/<name>`). It is a visual editor for printed documents: elements are placed on the page in millimetres, bound to saved datasets, previewed, and saved.

Four facts explain most of its behaviour:

- **The page is measured in millimetres.** Positions, sizes, margins and row heights are physical measurements. A report composed on a laptop prints the same in the plant office.
- **The preview is the report.** Preview sends the document to the server, which composes and renders it with the same composer a schedule uses.
- **Nothing is drawn from live data while you compose.** The canvas shows what each element is *bound to* — `Sum(Production.Weight)`, not a number — so that composing never loads the historian. To see numbers, preview.
- **A report is one format.** A report from the guided start, one started blank and one converted from sections are the same kind of document afterwards.

Starting a report

When the installation holds saved datasets, a new report opens with the **Start a report** strip across the top — "Answer three things and the pages are laid out for you. Everything is editable afterwards."



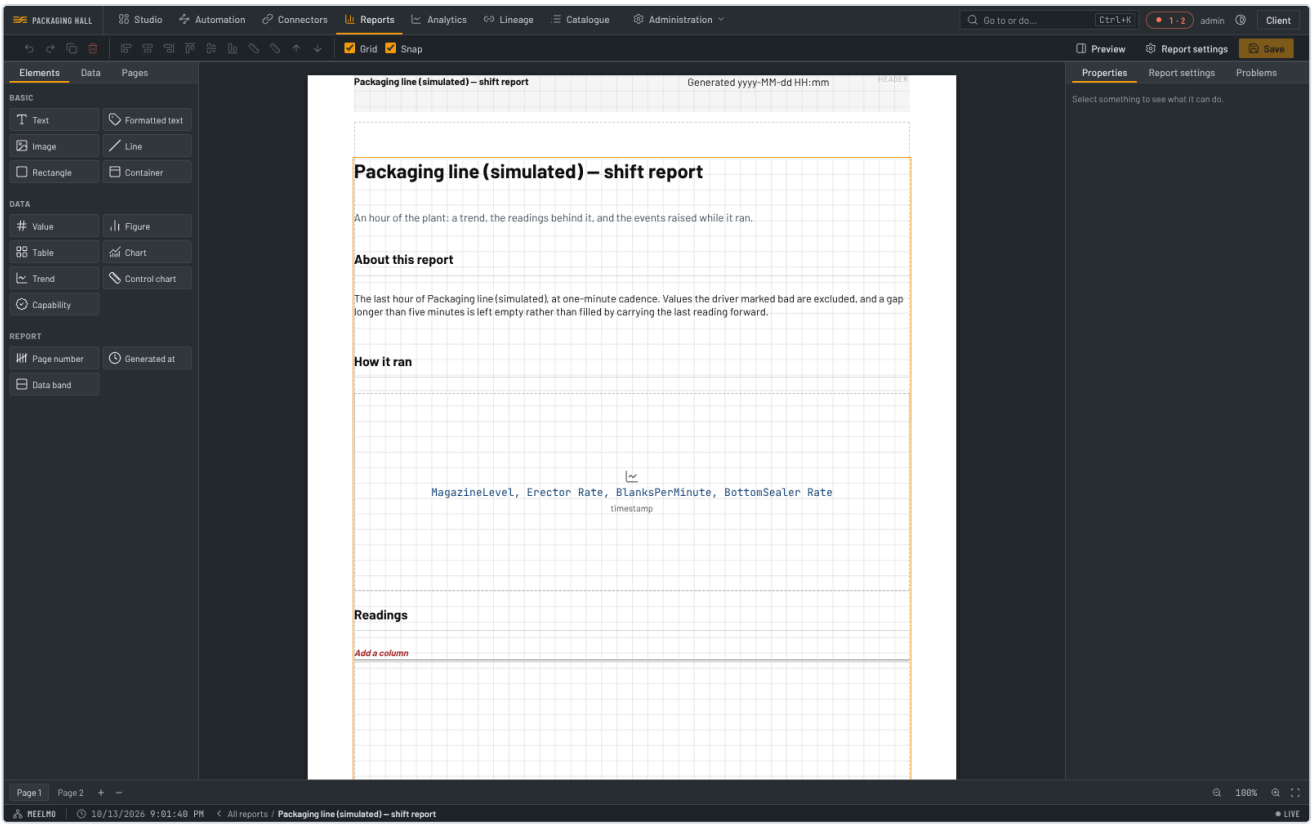
A new report in the Report Studio: the Start a report strip with Name, Title and Dataset on the left, What it shows with two of its three choices ticked, and Lay it out; the palette on the left and an empty A4 page with its header in the centre.

Field	Meaning
Name	The report's identifier, used in its address and in schedules (for example `shift-report`)
Title	What readers see at the top of the report (default **Shift report**)
Dataset	The saved dataset the report reads. Until one is chosen, **Lay it out** says "Choose a dataset first."
What it shows	Tick any of **The headline numbers**, **A table of the rows** and **A chart over time**; the counter shows how many are ticked (`2 of 3`)
Lay it out	Creates the pages from those answers
Blank page	Skips the guided start and leaves one empty A4 page

If the installation holds no datasets, the strip says "This installation holds no saved datasets to report on." Datasets are defined on the Desktop.

A new report is never quite empty: it arrives with one A4 page whose header carries the title and the generated stamp (`Generated yyyy-MM-dd HH:mm`) and whose footer carries the page number.

The workspace



The demonstration shift report open in the Report Studio: the header band with the title and generated stamp, the body with the title, the About this report note, the How it ran trend bound to four signals and the Readings table, and two page tabs at the bottom.

Where	What
Top bar	the product's destinations, as everywhere
Editing toolbar	**Undo**, **Redo**, **Duplicate**, **Delete**; the align group (**Align left**, **Centre horizontally**, **Align right**, **Align top**, **Centre vertically**, **Align bottom**, **Space out across**, **Space out down**); **Bring to front**, **Send to back**; **Grid** and **Snap** check boxes; on the right **Ask Assistant** (only where the Studio Assistant is configured), **Preview**, **Report settings** and **Save**. An **Unsaved** badge appears while there are changes nobody has saved
Left rail	**Elements** (the palette), **Data** (sources, datasets and parameters), **Pages** (the paper)
Centre	the page, white, at the current zoom, with the header, body and footer bands
Right rail	**Properties** (what is selected — "Select something to see what it can do." when nothing is), **Report settings**, **Problems**
Bottom	page tabs (**Page 1**, **Page 2**, **+** to add a page, **−** to remove this page), zoom out, the zoom percentage, zoom in and **Fit the page**
Status bar	the breadcrumb **All reports / <title>**

Elements

The palette has three groups. Drag an element onto the page, or click it to place it at the top-left of the band you are working in.

Group	Elements
Basic	**Text**, **Formatted text**, **Image**, **Line**, **Rectangle**, **Container**

Group	Elements
Data	**Value**, **Figure**, **Table**, **Chart**, **Trend**
Report	**Page number**, **Generated at**, **Data band**

Text and formatted text carry template text with `{{ }}` holes: `Shift report for {{ Parameter.Line }}` prints the parameter's value where the hole is.

Dragging from Data is the fastest way to compose:

Drag	Onto the page	Onto a table	Onto a chart
a dataset	a table with a column per field	—	—
a field	a figure bound to it, reduced	another column	another trace
a parameter	a figure bound to it	—	—

The reduction is chosen from the field: weights, energies, counts and totals are summed; a minimum is minimised and a maximum maximised; other numbers are averaged; text and timestamps take the last row. A timestamp dropped on a chart becomes its horizontal axis.

Data sources and parameters

Under **Data**, each data source lists its fields, with the installation's saved datasets below. **+** beside a dataset (**Add a data source**) reads it without placing a table. The pencil beside a source (**Change what this source reads**) opens its settings:

Setting	Means
Called	the identifier elements bind to (`Production` in `Production.Weight`); shown but not editable
Name	what a person reads
Dataset	the saved dataset it names
Rows at most	the source's own row ceiling
About the same runs as the report	whether the report's scope is applied to it (turn off for a lookup table)
Fields	which columns are kept; tick none to keep them all
Filter	rows kept when a field compares with a value or a parameter (**Add a filter**)
Group by, **Reduced per group**	one row per value of a field, with the reductions you add (**Add a reduction**)
Sorted by	the order, field by field (**Add a sort**, **Descending**)
Rows kept after sorting	a limit applied last
Remove	removes the source and clears the bindings that named it

Add process digest (under the tree) adds a source whose rows are the process digest over the report's window, one row per finding, most severe first. Instead of **Dataset** and **About the same runs as the report** it has **Limits of** (*Every dataset*, or one dataset's limits) and **Send when**, which holds a scheduled run back until the digest finds something that severe; the fields, filter, sort and limit work as for any source — a filter on `Level` ≥ 2 keeps warnings and worse.

Add alarm performance adds a source whose rows are the alarm-performance review over the report's window, one row per finding, worst first: `Rank`, `Verdict`, `Level` (3 unacceptable, 2 manageable, 1 acceptable, 0 for information), `Kind`, `Subject`, `Finding`, `Action`, `Magnitude`. It names no dataset; filter it like any source — `Kind` = `chattering` for the chattering alarms alone. Scheduled on the first of the month over the month before, it is the monthly alarm review ISA-18.2 asks a site to hold.

Add a parameter creates a value the report is run with. Its fields are **Name** (what an expression calls it, `Parameter.Line`), **What the person is asked**, **Type** (*Text*, *Number*, *Yes or no*, *Date*, *Date and time*, *One of a list*, *A place in the plant*), **Filled in by** (*Whoever runs the report*, *Always the default*, *The run's own window or scope*, *The clock*, *A data source's own values*), **Default**, **Required**, **Allow more than one**, and for a list, **Choices, one per line (value = label)**. A required parameter that is not supplied refuses the run rather than running the report over everything.

Properties of an element

The **Properties** tab groups what the selected element can do:

- **Position** — **Left**, **Top**, **Width**, **Height** (mm) and **Rotation**.
- **Data** — **Takes its value from**: *A fixed value*, *A field*, *A field, reduced* (**Reduced by** *Sum*, *Average*, *Lowest*, *Highest*, *Count of values*, *First value*, *Last value*…), *A parameter*, *Something about the run*, or *An expression*; plus **Number or date format**, **Before** and **After** (a prefix and a suffix, such as a unit) and **When there is no value**. **Reads as** shows the binding in words.
- **Appearance** — **Typeface**, **Size (pt)**, **Bold**, **Italic**, **Underline**, **Ink**, **Fill**, **Text alignment**, **Vertical alignment**, **Border (mm)**, **Border colour**, **Padding (mm)**, **Wrap long text**. A blank box means *inherited*.
- **When a condition holds** — **Only show it when** (an expression deciding whether it is drawn) and rules (**Add a rule**, **When**, **Hide it**) that change its colour or hide it; the last rule that holds wins.

A **Table** adds **Columns** (**Add a column**, with heading, format, width and **Total**), **Show the column names**, **Repeat them on every page**, **Add a totals row**, **Row height (mm)**, **Group by**, **Rows at most**, **Sorted by**, **Limits** (**Low**, **High**) and **Each row, when a condition holds**. A **Chart** adds **Kind** (*Line*, *Bars*, *Scatter*, *Distribution*, *Over time*), **Across**, **Plotted**, **On the right axis**, **A line per value of**, **Left axis label**, **Right axis label**, **Buckets** (distribution), **Draw a fit line** (scatter) and **Limits**. A **Data band** adds **Repeat once per row of**, **Start a group when this changes** and its bands (**Header**, **Above each group**, **Once per row**, **Below each group**, **Footer**).

Expressions use a small closed language: numbers, quoted text, `Source.Field`, `Parameter.Name`, `Context.GeneratedAt` (and `From`, `To`, `Title`, `Page`, `PageCount`…), the operators `+ - * / %`, comparisons, `and or not`, `? :`, and thirty functions such as `Sum`, `Avg`, `Round`, `If`, `Coalesce`, `Format` and `FormatDate`. The box says at once when it cannot read what you typed.

Pages

Pages in the left rail sets the paper: **Paper** (A4, A3, *Letter*, *Legal* or *Custom* with **Width (mm)** and **Height (mm)**), **Orientation** (*Upright* or *Sideways*), **Margins (mm)**, **Header height (mm)** and **Footer height (mm)**. **This page** changes only the page on screen; **Every page** changes the report's defaults, which every page without paper of its own takes; **Use the report's paper** gives a page back to the defaults. A band set to 0 mm is removed.

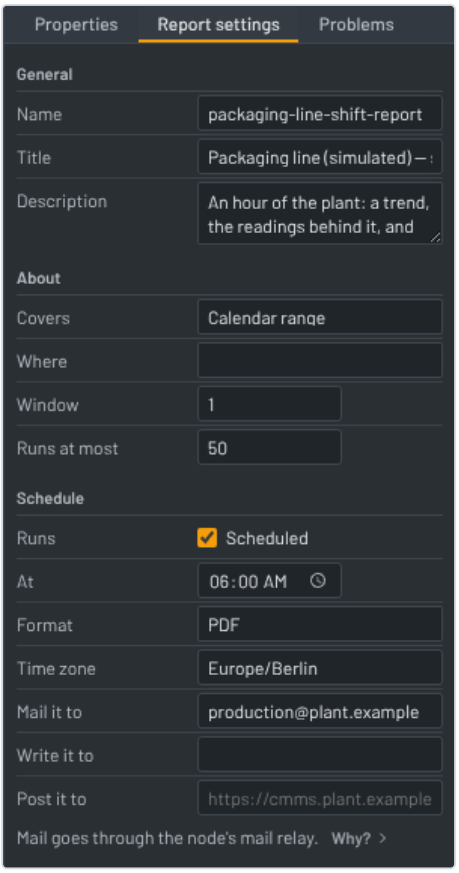
Only tables and data bands grow: they fill the body to its bottom and continue on a new page, with the column names repeated if the table says so. Headers and footers are stamped on every printed page; page numbers resolve after pagination.

Report settings, schedule and delivery

Report settings (toolbar, or the right-rail tab) holds the report's own facts:

Section	Field	Means
General	**Name**, **Title**, **Description**	identifier, printed title, and the line shown under the title in the list
About	**Covers**	*Calendar range*, *Each execution*, *Each state interval* or *Around each event*
	Of type	the run, event or state type (not shown for a calendar range), for example `order`
	Where	a place in the plant's asset tree, to narrow the report to one area
	Window	the default look-back in hours (1 to 8760)
	Runs at most	how many runs the scope may cover (1 to 1000)
Schedule	**Runs**	tick **Scheduled** to make the report run by itself
	At	the time of day
	Format	*Web page*, *PDF* or *Excel*
	Time zone	the plant's time zone as an IANA id, for example `Europe/Berlin` (default `UTC`); an unknown id is flagged
	Mail it to	recipients
	Write it to	a directory on the node

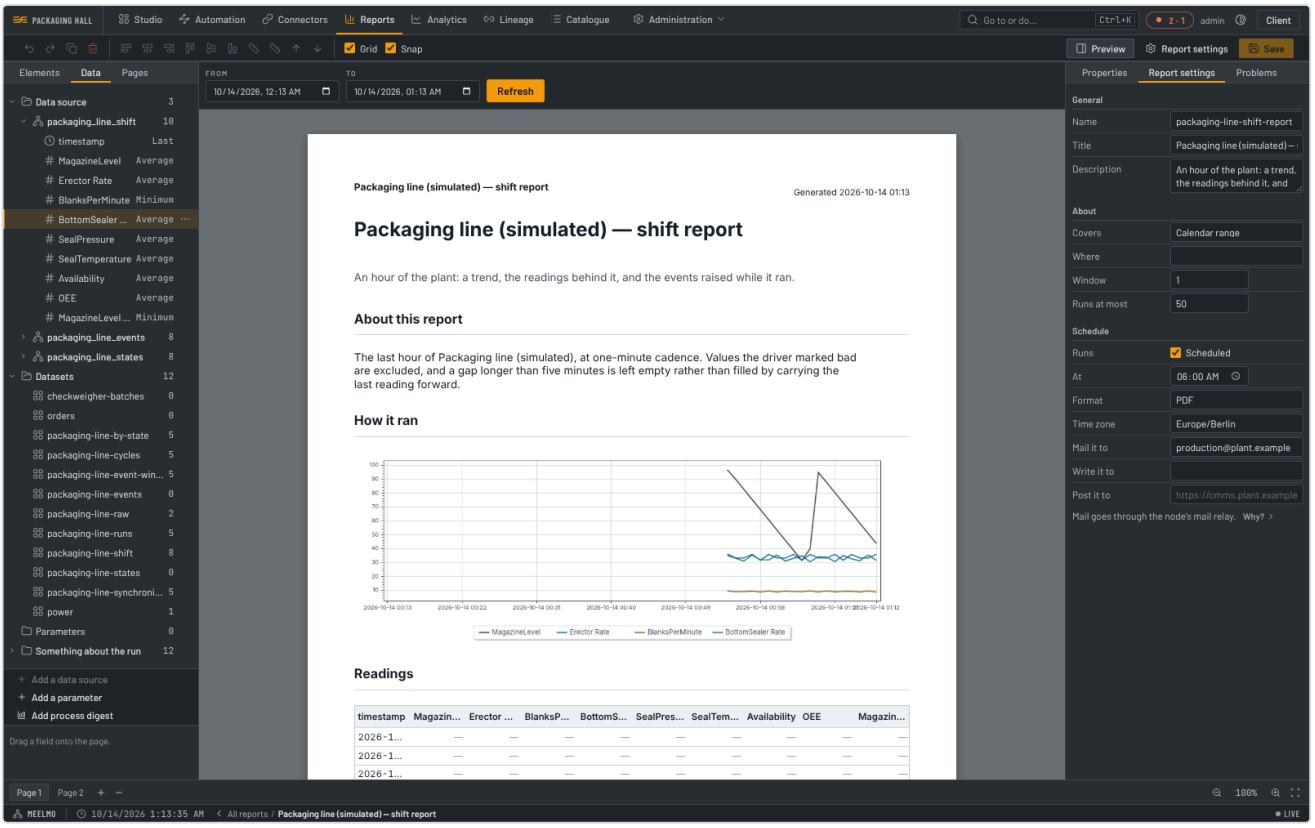
The **Schedule** section is shown only to someone who holds `Reports.Schedule`; anyone else reads "You are not permitted to compose reports." in its place. A report names only *who* it is mailed to: the mail goes through the node's mail relay, which the Desktop sets under **Acquisition ▸ Alarms ▸ Notifications** ("Mail goes through the node's mail relay."). A node with no relay still writes the report to disk. After an outage, a schedule catches up with at most the four most recent missed runs and reports the rest as skipped.



Report settings of the shift report in the right rail: General, About with Calendar range, a window of 1 and at most 50 runs, and Schedule with Scheduled ticked, 06:00, PDF, Europe/Berlin and production@plant.example under Mail it to, then the line saying mail goes through the node's mail relay.

Preview, Problems and Save

Preview splits the view and shows the report as the server composes it, over the window in its own **From** and **To** boxes; **Refresh** runs it again ("The preview is the report: same data, same renderer, same pages."). **Problems** lists what the validator refuses — a binding to a field the dataset does not produce, a chart without a horizontal axis, a parameter renamed while expressions still use the old name — or says "Nothing to fix.". **Save** stores the report; a report that shows no problems will save.



The shift report previewed in the Report Studio: From and To over the last hour with Refresh, and the page as the server composed it — the header and generated stamp, About this report, the How it ran trend and the start of the Readings table; the Data tree on the left and Report settings on the right.

A report composed of sections (in the Desktop's form, or over the API) opens exactly as it is. **Turn sections into pages** converts it into elements on A4 pages in one undo step, before you save.

The Report Studio shares its keyboard with the HMI Studio: `Ctrl+Z` / `Ctrl+Y` undo and redo, `Ctrl+C`, `Ctrl+X`, `Ctrl+V`, `Ctrl+D` duplicate, `Delete`, arrows move by 1 mm and `Shift`+arrows by one grid step, `Ctrl+]` / `Ctrl+[` change the draw order, `Ctrl+=` / `Ctrl+-` / `Ctrl+0` zoom, `Escape` cancels a drag or clears the selection.

Worked example: a scheduled report of the erector rate

The line manager wants a one-page report of the erector's rate over the last day, e-mailed as a PDF every morning at 06:00 plant time.

1. **Reports ▸ New report**. The **Start a report** strip appears.
2. **Name** `erector-daily`, **Title** `Erector — last 24 hours`, **Dataset** `packaging-line-shift`.
3. Under **What it shows** tick **The headline numbers** and **A table of the rows**; leave **A chart over time** unticked. Click **Lay it out**.
4. In the left rail open **Data**, expand the source and drag the `Erector Rate` field onto the page beside the headline figures. The figure is bound to the field's average; in **Properties ▸ Data** set **After** to `cases/min`.
5. Select the table. Under **Limits**, **Add a limit** on the `Erector Rate` column with **Low** `20` and **High** `30`, so the minutes that left that band are marked.
6. Open **Report settings**. Under **About** set **Window** to `24` hours.
7. Click **Preview**, set **From** to yesterday 06:00 and **To** to today 06:00, and check the numbers and the coverage note.
8. Under **Schedule** tick **Scheduled**, set **At** `06:00`, **Format** *PDF*, **Time zone** `Europe/Berlin`, **Mail it to** `production@plant.example`.
9. Read **Problems** — it should say "Nothing to fix." — and click **Save**.

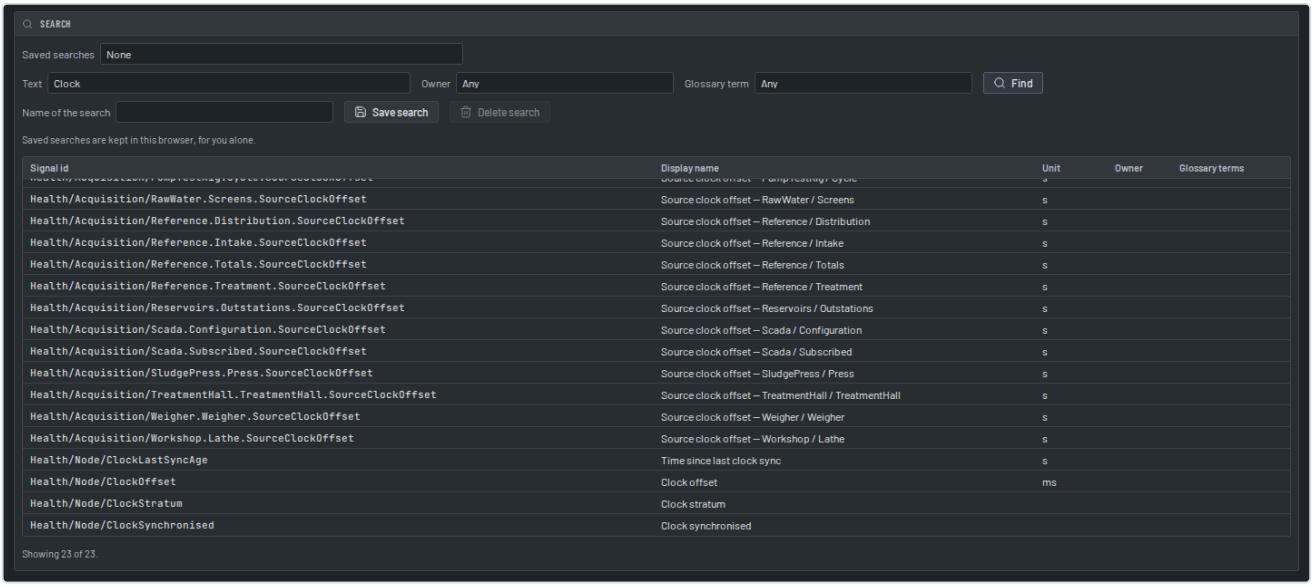
10. Back on **Reports**, the **Scheduled** folder lists the report with **At** `06:00`, **Format** *PDF* and **Goes to** `production@plant.example`.

The signal catalogue

Catalogue in the top bar opens `/catalogue`, for everybody: the route behind it searches only the signals your session may read.

Search. Text matches a signal's id, display name, description or unit, and the glossary terms, synonyms and owner linked to it — so *skin temperature* finds the pyrometer whatever its tag is called. **Owner** and **Glossary term** narrow to one of each, matched whole. **Find** asks the node, so the answer is the one the Desktop, the data API, Grafana and an assistant would get. The table shows each signal's owner and terms beside it, and how many matched.

The clock, as signals. The node publishes its clock synchronisation as health signals — `Health/Node/ClockOffset`, `ClockSynchronised`, `ClockStratum` and `ClockLastSyncAge` — and the offset between each acquisition group's device time and its arrival as `Health/Acquisition/<connection>.<group>.SourceClockOffset`. Searching *Clock* finds them all; they trend, alarm and publish like any other signal. They detect drift; they do not set the clock.



The catalogue searched for Clock on the reference works: 23 signals, the SourceClockOffset of each acquisition group in seconds, then the node's own ClockLastSyncAge, ClockOffset, ClockStratum and ClockSynchronised

Saved searches. Type a **Name of the search** and press **Save search**: the text, owner and term are kept under that name in **Saved searches**, in this browser and under the name you signed in with. Choosing one runs it; **Delete search** forgets it. A saved search is yours and is not configuration.



The Catalogue page on the reference works: the saved search Network pressure finding four signals through the Supply pressure term, each with its owner; the glossary of five terms with Supply pressure loaded into the form below; and the owners and steward notes

Glossary. The terms the project holds, each with its definition, synonyms and how many signals it is linked to. Press a row to load it into the form below: **Term**, **Definition**, **Synonyms** (one per line or separated by commas), **Linked signals** and **Linked assets**, one id per line; tick **Take this term out of the project** to remove it. **Put it into the project…** writes that one term into the node's configuration draft, as you, and shows what activating the draft would change — exactly as keeping a model or a rule from an analysis page does — and **Activate** applies it, with a sentence for the revision history. You may link only signals you may read; a term's links you cannot see are kept as they are, and a term that has any is not yours to remove.

Owners and steward notes. The same, for one signal or asset at a time: **About**, the **Signal or asset id**, the **Owner** and the **Steward notes**. A blank owner and blank notes take the entry out. An administrator — who may list the node's users — is offered those users and their roles as suggestions in the **Owner** box; anybody may still type an owner who is not on the node, such as a team mailbox, and the page says so. A session that may not list the node's users is offered nothing and is told to type the owner.

A session the node does not let change configuration sees the glossary and the owners and is told it cannot put a change into the project. A host that cannot read the node's glossary says so above the results, and its search then matches the signals' own fields only.

Lineage

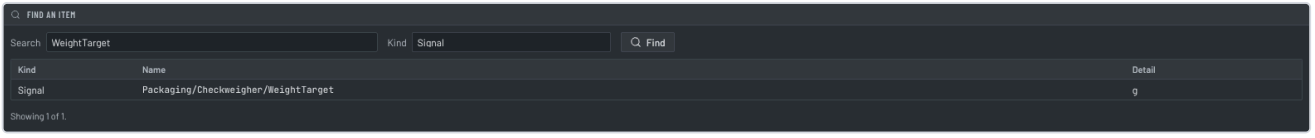
Lineage in the top bar opens `/lineage`, for everybody. It draws where an item comes from and everything inside Meelmo that depends on it. It is the browser's view of *Acquisition and I/O configuration ▸ Lineage*. The drawing is computed from the configuration the node is running, each time you ask, and

each person sees only the part of the plant they may read. It follows data as far as it leaves the node (a dataset's drops and OData feed, a publication); it does not follow it into the systems downstream of Meelmo.

Finding an item

Field or control	Meaning
Search	Part of a name: a tag, a signal, a rule, a screen...
Kind	**Any kind**, or one of **Connection**, **Tag**, **Signal**, **Computed signal**, **Alarm**, **Recording trigger**, **Detection rule**, **Model**, **Flow**, **Screen**, **Dataset**, **Dataset drop**, **Report**, **OData feed** and **Publication**
Find	Lists what matches, with its **Kind**, **Name** and **Detail**, and "Showing N of M." under the table. Press a row to open it

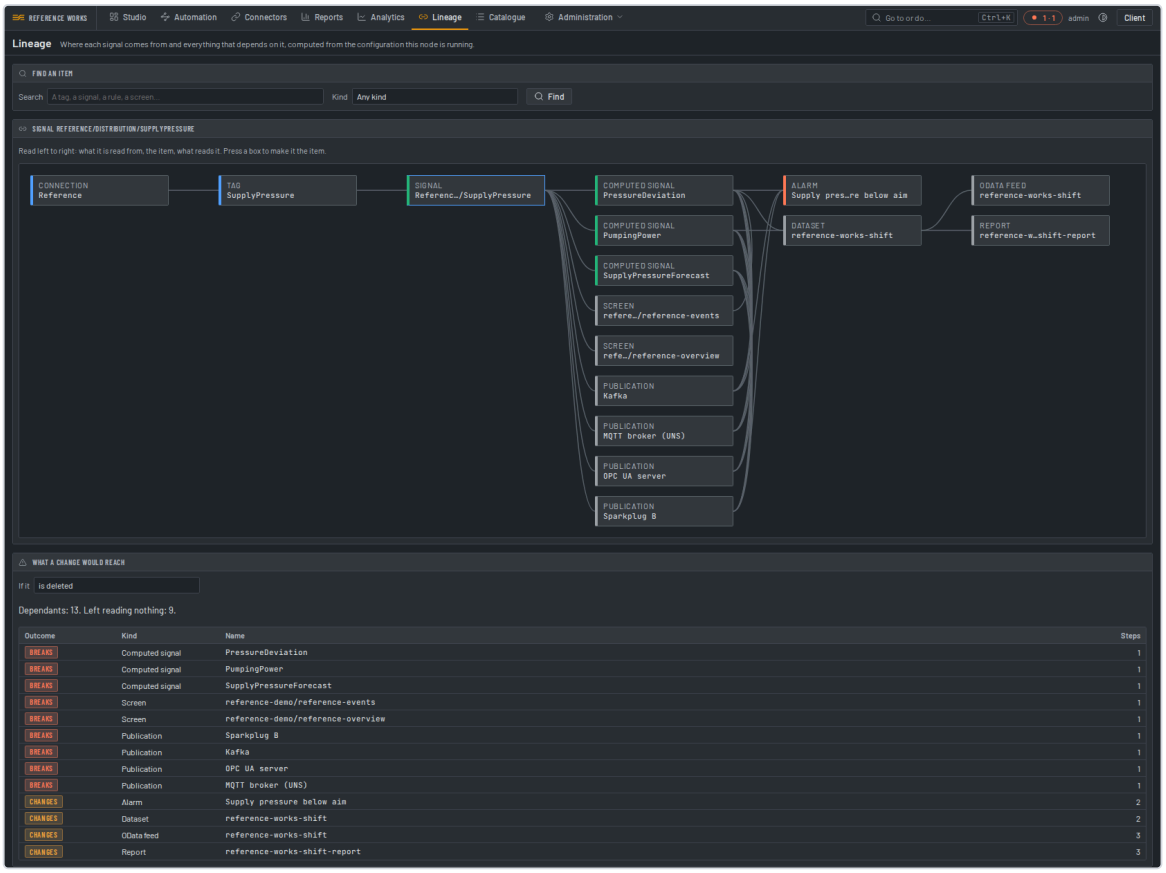
Nothing in this node's configuration matches. means the search found nothing. A configuration larger than a lineage is computed for is cut, and the page says so.



Find an item with WeightTarget searched for among signals: one match, the signal Packaging/Checkweigher/WeightTarget with its unit g, and Showing 1 of 1.

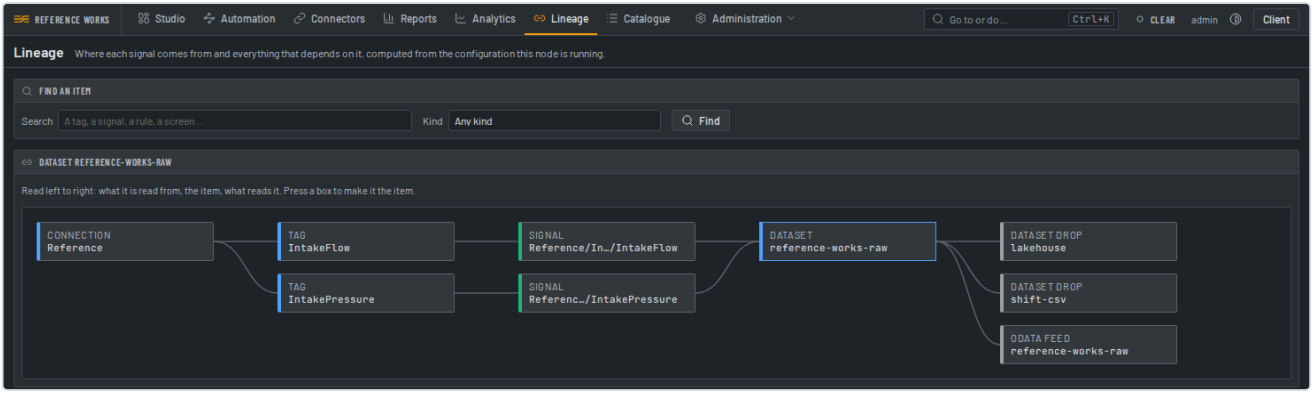
Reading the drawing

The panel is titled with the item's kind and name. Read it left to right: what the item is read from, the item itself, and what reads it. Press any box to make it the item; the address changes to `/lineage?item=…`, so a lineage can be sent as a link and the browser's **Back** returns to the previous item. A column with more boxes than fit ends in **+N more**. When only the nearest part is drawn, the page says so.



Lineage of the signal Reference/Distribution/SupplyPressure: from the Reference connection and its tag, to the computed signals, the screen, the alarm Supply pressure below aim, the dataset, its OData feed and the report; and What a change would reach if it is deleted — three dependants break, four change

A dataset's lineage reads to the right as far as it leaves the node: its drops to a lakehouse path and its OData feed.



Lineage of the dataset reference-works-raw: the Reference connection, the tags IntakeFlow and IntakePressure and their signals on the left; on the right the two drops of the dataset, lakehouse and shift-csv, and its OData feed

What a change would reach

For a connection, a tag, a signal or a computed signal, the panel **What a change would reach** lists every dependant of one change, nearest first.

Field or column	Meaning
If it	The change to ask about: **is deleted** (the default), **is renamed**, **changes type**, **moves address** or **is switched off**

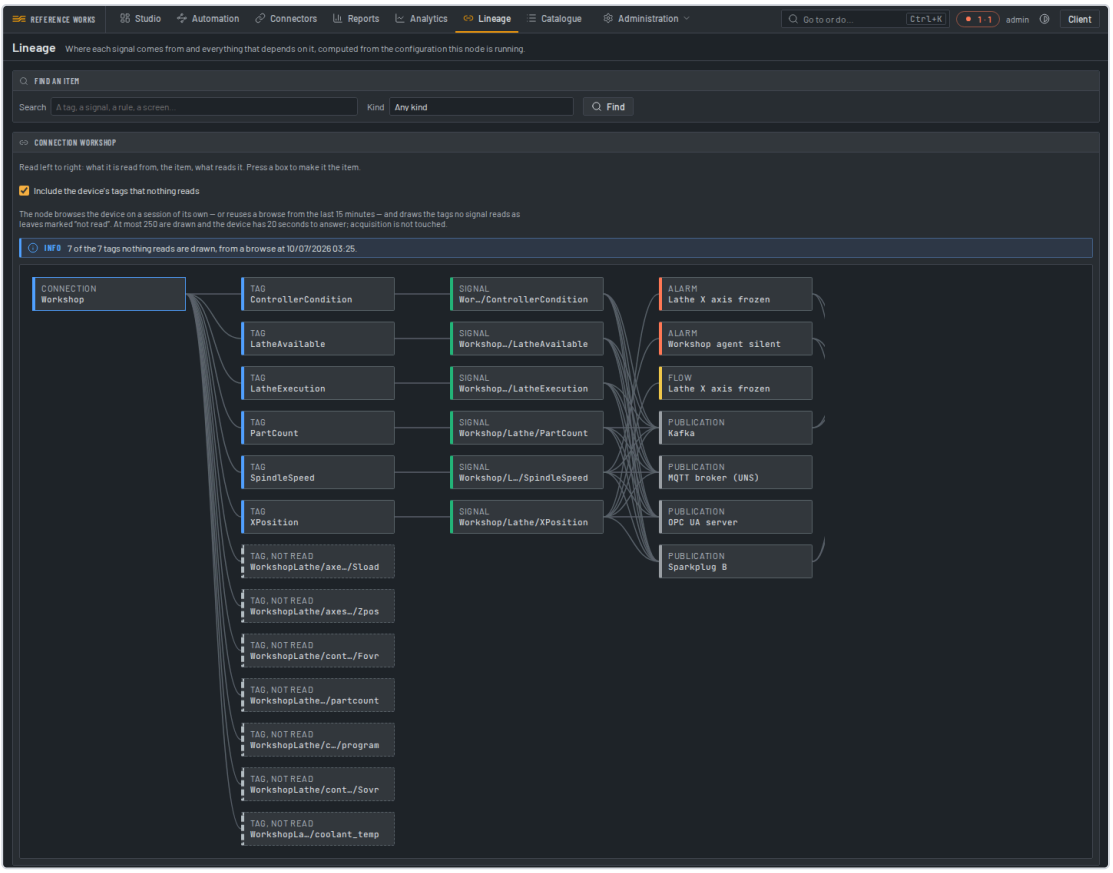
Field or column	Meaning
The line above the table	"Dependants: N. Left reading nothing: M.", or *Nothing depends on it.*
Outcome	**Breaks**: the dependant would be left reading nothing. **Changes**: it keeps reading, but something under it has changed
Kind, **Name**	The dependant
Steps	How far it is from the item

The device's tags that nothing reads

A lineage is computed from the configuration, so it cannot show a point in the controller that nobody has configured. For a connection, tick **Include the device's tags that nothing reads**. The node browses the device on a session of its own, or reuses the tag-drift browse when one is less than 15 minutes old. It draws every tag no signal reads as a dashed leaf marked **Tag, not read**. At most 250 are drawn, the device has 20 seconds to answer, and acquisition is never touched. A banner under the box says what happened:

Banner	Meaning
"N of the M tags nothing reads are drawn, from a browse at ..."	The browse worked
Every tag the device offers is read by a signal.	Nothing is unread
This connection's driver cannot list its device's tags, so the tags nothing reads cannot be shown.	The driver has no browse
You are not permitted to read any of this connection's tags, so its device is not browsed.	An account the data boundary narrows sees a device's unread tags only when it may read at least one of that connection's tags
This server does not hold the node, so it cannot browse its devices.	The host has no node of its own to browse with
"The device could not be browsed: ..."	The device did not answer, with the reason

A browse that reached the device is recorded in the audit as `api.lineage.browse`.



The Workshop connection with Include the device's tags that nothing reads ticked: its six configured tags and their signals, and seven dashed leaves for the data items the lathe's agent offers that no signal reads

On a plant server, where the recorder service holds the node and this web host serves it, Lineage works the same way: the host reads the lineage the node computes and draws each person only the part their account may read; the node browses a device for its unread tags. A server attached to no node says so in the page's banner instead.

Worked example: before renaming the checkweigher's target

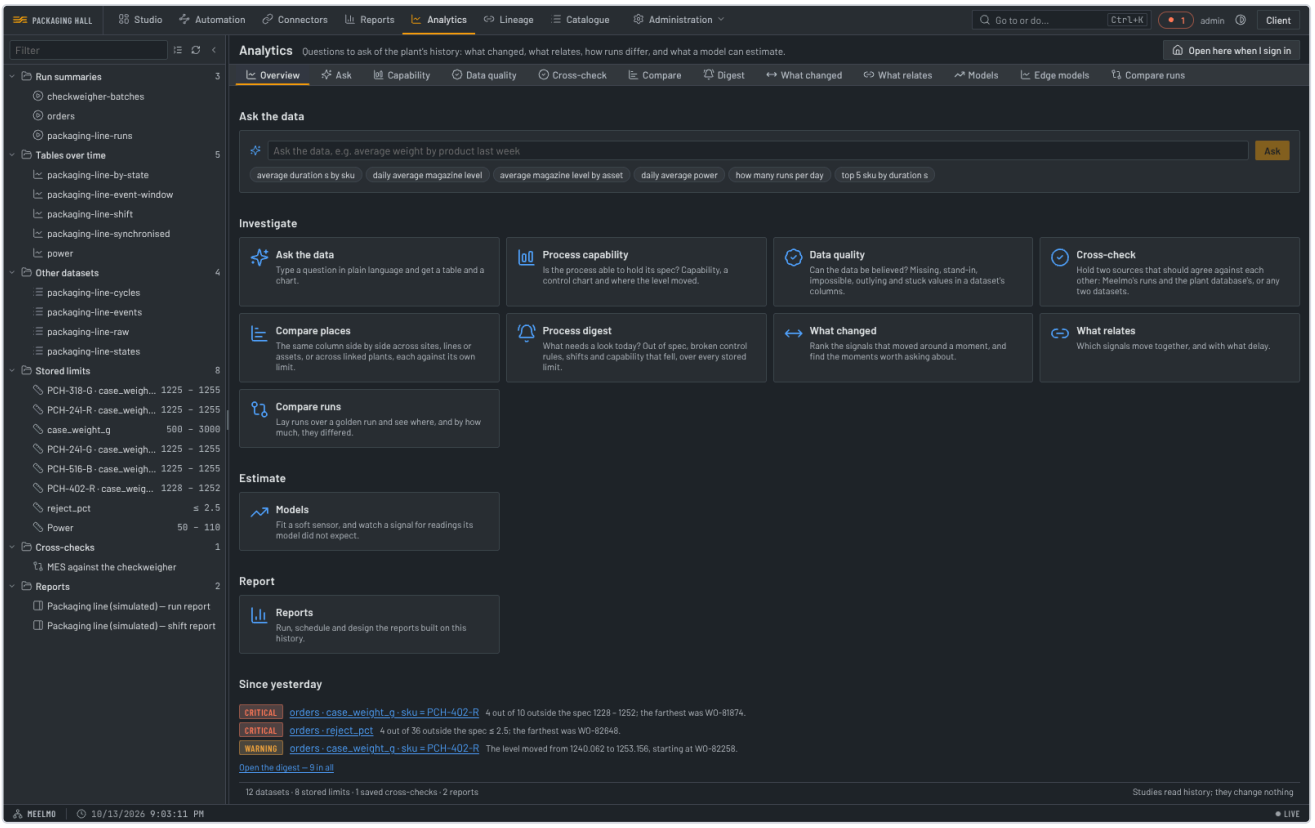
1. Press **Lineage** in the top bar.
2. Type `WeightTarget` in **Search**, choose **Signal** under **Kind**, and press **Find**.
3. Press the row `Packaging/Checkweigher/WeightTarget`. The drawing shows the `Packaging` connection and its tag on the left and everything that reads the signal on the right.
4. Under **What a change would reach**, choose **is renamed** in **If it**. Each row marked **Breaks** is something that would read nothing after the rename and has to be changed with it.
5. Press the `Packaging` box to make the connection the item, and tick **Include the device's tags that nothing reads** to see what else the device offers.

Analysis in the browser

Analytics in the top bar opens `/analytics`, which starts with a question box — type a question in plain language and it is answered on the **Ask** page — and then lists the tools grouped by the question they answer — **Investigate**, **Estimate** and **Report** — each with one line on what it is for. A row of tabs across every analysis page moves between the questions without going back to a menu; the first, **Overview**, returns to the hub:

Tab	Address	Question
Overview	`/analytics`	What can I ask here?
Ask	`/analytics/ask`	A question in plain language — see Ask the data
Capability	`/analytics/process`	Can the process hold its spec, and is it behaving as it usually does?
Data quality	`/analytics/quality`	Can the numbers in a dataset be believed at all?
Cross-check	`/analytics/crosscheck`	Do two sources that should agree actually agree?
Compare	`/analytics/compare`	How does the same value compare across the sites, lines or assets of this plant?
Digest	`/analytics/digest`	What did the process do since the last period that somebody should hear about? — see Process digest
Alarms	`/analytics/alarms`	Which alarms are nuisances, and how loud was the alarm system? — see Alarm performance
What changed	`/investigate`	Which signals were doing something different when it happened?
What relates	`/investigate/relations`	Which of these signals have anything to do with each other?
Models	`/models`	Belongs to the Models module
Edge models	`/models/edge`	Belongs to the Models module: the trained models running at the edge
Compare runs	`/analysis/runs`	How did this run differ from the good ones?

Each tab is its own address, so "have a look at what relates on the checkweigher" is a link somebody can send. Only the tabs this session can use are drawn: every tab from **Ask** to **Edge models** needs an account that may read history, and **Overview** and **Compare runs** are drawn for everybody. When only one tab would be left, the row is not drawn.



The Analytics overview on the packaging node: the explorer on the left with the run summaries orders and checkweigher-batches, the tables over time, the eight stored limits, the saved cross-check and the two reports; on the right the Ask box with example questions made from the datasets, the tools grouped under Investigate, Estimate and Report, and Since yesterday listing the three most severe findings of the last day, led by PCH-402-R's case weight out of spec, each a link to its study, with Open the digest.

The Analytics workbench

Every analysis page sits in one workbench, laid out as REPXplore's desk is:

- **The explorer** on the left lists what the analyses are made of: **Run summaries** (datasets with one row per run), **Tables over time** (one row per timestamp or interval), **Other datasets** (event, state and sequence tables — listed so nobody wonders where they went, but not something a capability study reads), **Stored limits**, the saved **Cross-checks**, and the **Reports** this person may view. Clicking a dataset opens it in **Process capability**; clicking a stored limit opens the study already filtered to what the limit is for; clicking a cross-check opens it on the **Cross-check** page and runs it; clicking a report opens it. The filter box, collapse-all and refresh work as in the Reports area, the splitter can be dragged, and the explorer can be folded away.
- **The page** on the right: its title with one line saying what it asks, the tabs, and the page itself.
- **The status line** at the foot says how many datasets, stored limits, saved cross-checks and reports the explorer holds, and that studies only read history.

In your language. The sentences an analysis writes about its answer — a study's notes and warnings, the steps of *How was this computed?*, a quality check's verdict and the reason for each finding (freshness, gaps at the edges of the period and thin days included), a cross-check's and a comparison's notes, why the digest could not judge a limit, and the question planner's plan, choices and refusals — are shown in the language you chose, with numbers, dates and durations written your way. So are the dataset's own warnings (a segmentation that matched no window, the row limit, runs too long to summarise), the Models page's warnings and a tuning sweep's reservation, the sentence under a refusal's headline when an analysis turns a request down (no such dataset, a column or field it does not have, a shape it does not read, a permission you lack, a pick the answer is not grouped by), and the page's own small words: *(none)*

for items with no value of the field grouped by, *typed in* for limits you entered, a planner's *day*, *period* or *count* column. The server sends each sentence in English for scripts and assistants, and beside it the same sentence in a form the page can say again in any of the product's languages.

What stays in English, and why. A sentence shown in English is marked as English so a screen reader reads it with an English voice. Three kinds remain:

- **A linked plant, or a node, on an older release.** It sends its sentences in English only, with no key beside them, and the page shows them exactly as that plant wrote them. This is expected: the page never guesses a translation of a sentence it was not given the key for, and once that plant is upgraded its sentences are shown in your language like the rest.
- **Words that did not come from Meelmo.** An exception from a driver, the historian or the runtime, the reason a regular expression is invalid, the notes an assistant writes beside a shortlist on the Models page — these are passed on as they were written.
- **Checks on a request's own fields** that a page cannot send — a model document with no id, a lag that reads the future, a dataset definition the Datasets editor would not have saved. A script or an assistant meets these, and reads English.

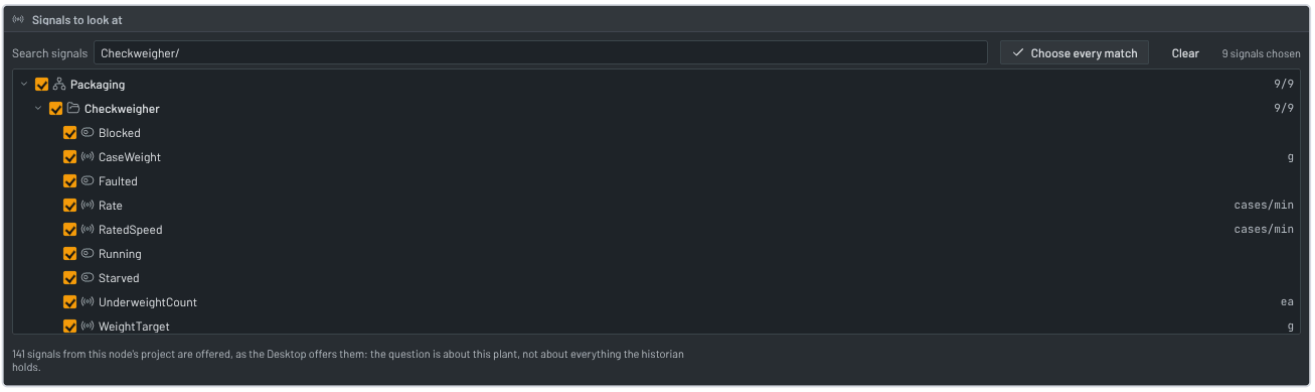
Answers saved as files. *MD* writes the question, the *Planned:* line, the column headings and the warnings in your language, as the page shows them. *CSV* and *JSON* keep the server's own column names (`day` , `count` , `CaseWeight__avg`) and, in JSON, its English plan and warnings, whatever your language: a spreadsheet's formulas and a script's keys must read the same file from every desk.

Choosing the signals

Both **What changed** and **What relates** start with the **Signals to look at** panel.

Control	Does
Search signals	Filters the tree by part of a name or path, for example `Checkweigher/`
Choose every match	Ticks every signal the search matched — "everything under Checkweigher" is two keystrokes
Clear	Unticks everything
N signals chosen	How many are ticked
The tree	The node's project signals by connection (`Packaging` , 113 signals in the demonstration) and the virtual signals (`Virtual` , 28), each branch with its count. Tick a branch to take everything under it

The note under the tree says what is offered: "141 signals from this node's project are offered, as the Desktop offers them: the question is about this plant, not about everything the historian holds." Where the host cannot say which signals the project acquires, every signal in the catalogue is offered and the note says so. The licence limits how many signals one question may take; over the limit the page says "The license permits N signals in one question…" and asks you to narrow the choice.



Signals to look at with Checkweigher/ searched for and Choose every match pressed: the nine checkweigher signals ticked under Packaging, 9 signals chosen, and the note that 141 signals from this node's project are offered.

Ask the data

Analytics ▸ Ask the data (the **Ask** tab, `/analytics/ask`) answers a question typed in plain language: *average case weight by SKU last week*, *how many orders per day this month*, *top 5 products by max seal temperature*. It is REPXplore's **Query** page. The same box is the first thing on the **Analytics** overview; a question typed there opens this page with it.

Type the question and press Enter (or **Ask**). The chips under the box are example questions made from your own datasets; each has been checked to plan against the dataset it was made from, and clicking one asks it of that dataset.

How the question was understood. The blue **Planned** line comes before the answer and says what the question was taken to mean: the dataset, what is measured and how (*average CaseWeight__avg*, *count of runs*), grouped by what, the order and how many rows, the filters, and the period with its dates. No AI model reads the question: its words are matched to the saved datasets' own names, columns, fields and the values their fields hold, so the same question always gets the same plan, and the **Planned** line is the whole of what happened to it.

What a question can say. The planner knows no plant's words; it knows your datasets. A column `Seal_Temperature__avg` answers to "seal temperature" (the `__avg`, `__max`, `__min`, `__sum` endings are how it was summarised over each run, not part of the name), a context field `sku` to "sku" or "skus", `execution_type` to "execution type".

You want	Write
A breakdown	`average weight by sku`, `max pressure per product`
A ranking	`top 5 skus by weight`, `bottom 3 products by energy`
The runs themselves	`top 10 runs by max weight` — each run named by its identifier
A count	`how many runs by status`, `number of runs per day`
A trend	`daily average pressure`, `weekly total gas`, `how many runs per hour`
A filter	`sku = PCH-241-R` (quote values with spaces: `line = "Line 2"`), `CaseWeight__max > 1300`, or name a value the data holds: `how many completed runs by product`, `average energy for product A`
A period	`today`, `yesterday`, `last week`, `this month`, `the last 7 days`, `in March`, `in 2025`, `since 2026-07-01`, `on 2026-07-15`, `2026-07-01 to 2026-07-15`

Words for how values are summed up are *average*/*mean*/*avg*, *max*/*maximum*, *min*/*minimum*, *total*/*sum*, *count*/*number of*/*how many*; for grouping *by*, *per*, *each*, *daily*, *weekly*, *monthly*, *hourly*; for ranking *top N*, *bottom N*, *highest*, *lowest*. Asked for the maximum, a run summary's `__max` column is used and the largest of those taken; asked for the total, its `__sum` column where it has one; otherwise its `__avg` column. Without a period, the dataset's own window is used; relative periods count your days, in your browser's time zone, and weeks start on Monday. At most three filters and two groupings (a field and a period, or two fields).

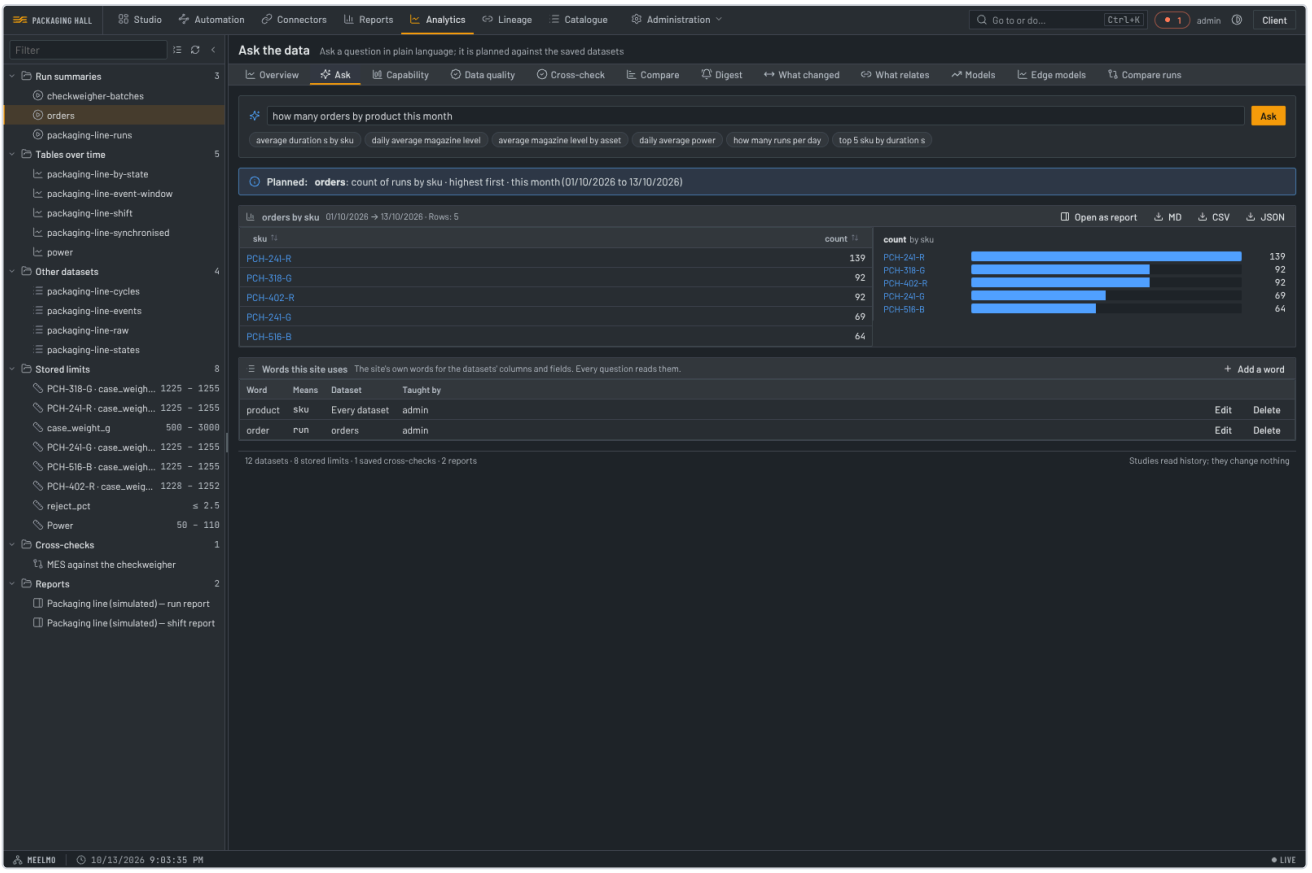
Questions may be asked in **Italian** or **Spanish** as well: *peso medio per sku negli ultimi 7 giorni*, *cuántas ejecuciones por día el mes pasado*. The phrasing, the periods, the rankings and the words for summing up are read as English, names and filter values are kept as written, and a grey **Read as** line shows how the question was read. The same question gets the same plan in every language.

When it is unsure, it asks. A question two datasets explain equally, a word that is part of two columns' names (*temperature*, in a dataset of top and bottom sealer temperatures), or an *average* of nothing named comes back as **Which did you mean?**: each choice is a question that means one thing, and clicking it asks it. Nothing is run until you choose.

Teaching the site's words. A word named back can be taught. Somebody who may change datasets (who designs reports) sees **Teach this word** beside each word the question could not place: choose what it means — a column (*CaseWeight*), a field to group or filter by (*sku*), or each run of a run summary — and whether it means that in the dataset the question came closest to or in every dataset, then **Save**. The question is asked again straight away. From then on every question reads the word the way the dataset's own names are read, plurals and accents aside: taught *product* = *sku*, *how many orders by product* plans as *count of runs by sku*; taught *order* = each run of `orders`, *orders* counts them. A word taught for one dataset wins there over the same word taught for every dataset.

The words are listed under the answer in **Words this site uses**, with what each means, where, and who taught it; **Add a word**, **Edit** and **Delete** are there for the same people. On a node the words are configuration beside the datasets, as the stored limits are: teaching one activates a revision, so it can be traced and rolled back. A word must mean something a dataset has, so a typo is refused rather than kept.

When a word means nothing. Every word of a question must mean something in the dataset that answers it. A word no dataset explains is named back — "*seal integrity*" *is not something the datasets measure or group by* — with what the closest dataset does measure and group by, and nothing is answered: an answer that ignored the word would answer another question. Before refusing, the page reads the closest dataset's rows once and tries again with the values its fields hold, so *completed* is understood as `status = Completed` without anyone typing it. It also refuses forecasts and causes (*why…*, *predict…*), which recorded rows cannot answer, and thresholds said in words (*above 1300*): write them as a filter on the column, `CaseWeight__max > 1300`.



Ask the data answering how many orders by product this month from the orders dataset: the blue Planned line — count of runs by sku, highest first, this month with its dates — then the five SKUs with their counts beside a ranked bar chart, and under the answer the two words this site taught, product meaning sku in every dataset and order meaning each run of orders.

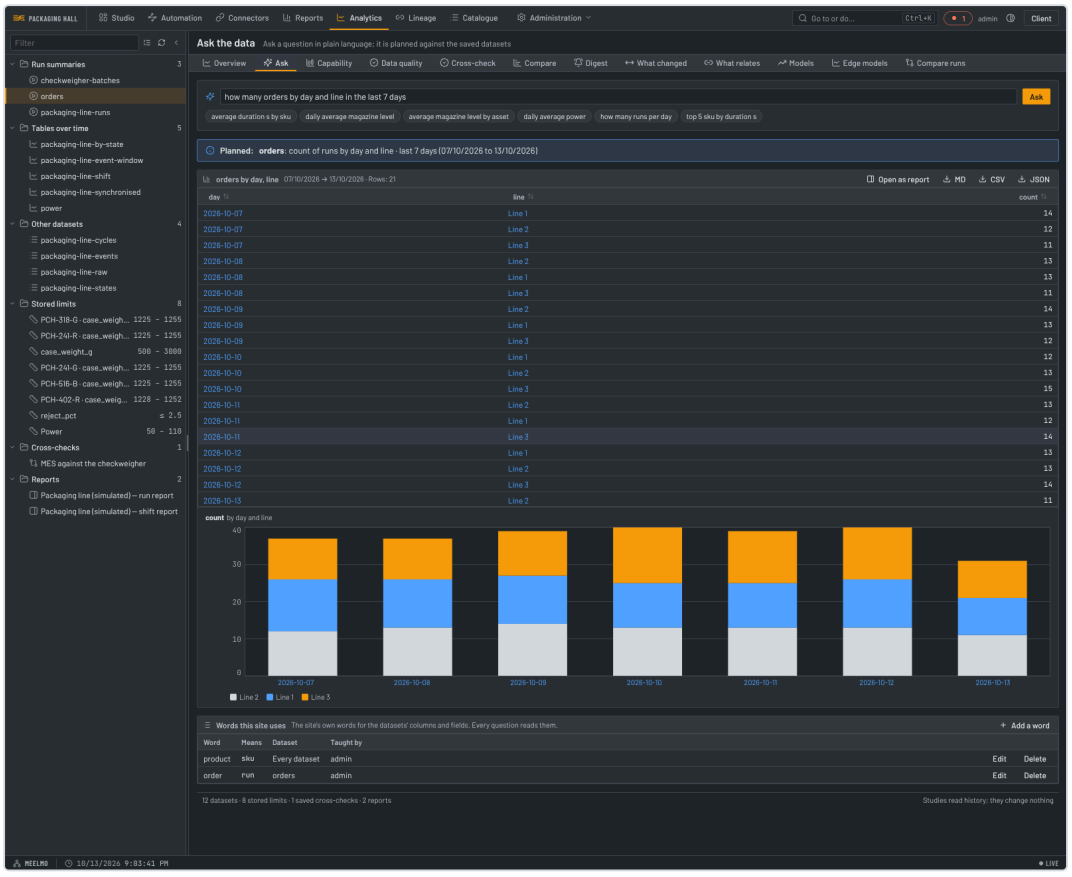
The answer is a table with, beside it, the chart the answer suggests — ranked bars for categories, a line over days (a few columns when there are only one or two). The table and the chart sit side by side up to twenty categories and one above the other past that; the chart draws at most thirty bars, the table every row, up to 500. Warnings above the answer say what makes it smaller than it looks: rows the dataset stopped at, no rows in the period. **MD**, **CSV** and **JSON** at the top of the answer download it — Markdown with the question and the plan for a note, CSV with plain numbers for a spreadsheet, JSON with the plan and the rows for a script.

Column names. A measured column is named after what it measures. Over a plain column the aggregate is a suffix, as in a Report Studio query: *average pressure per day* gives `Pressure_avg`. A run summary's column already says how each run was summarised, so the answer's aggregate over the runs is put in front instead: *average CaseWeight__avg by sku* gives `avg of CaseWeight__avg` (not `CaseWeight__avg_avg`), *max weight by sku* gives `max of CaseWeight__max`. A count is `count`. The CSV and JSON downloads use the same names.

Two groupings are drawn too. *How many runs by day and product*, *average weight by sku and line*: the first grouping runs along the axis (the period, when one of the two is a period) and the second is drawn as series, one colour each, with a legend under the chart and each category named under its columns. A count or a total is stacked, because its parts add up to the category's whole; an average, a highest or a lowest is set side by side, because stacking them would draw a sum nobody computed. The chart draws the eight largest series and the first twenty categories; when it leaves some out it says so under the legend, and the table still has every row.

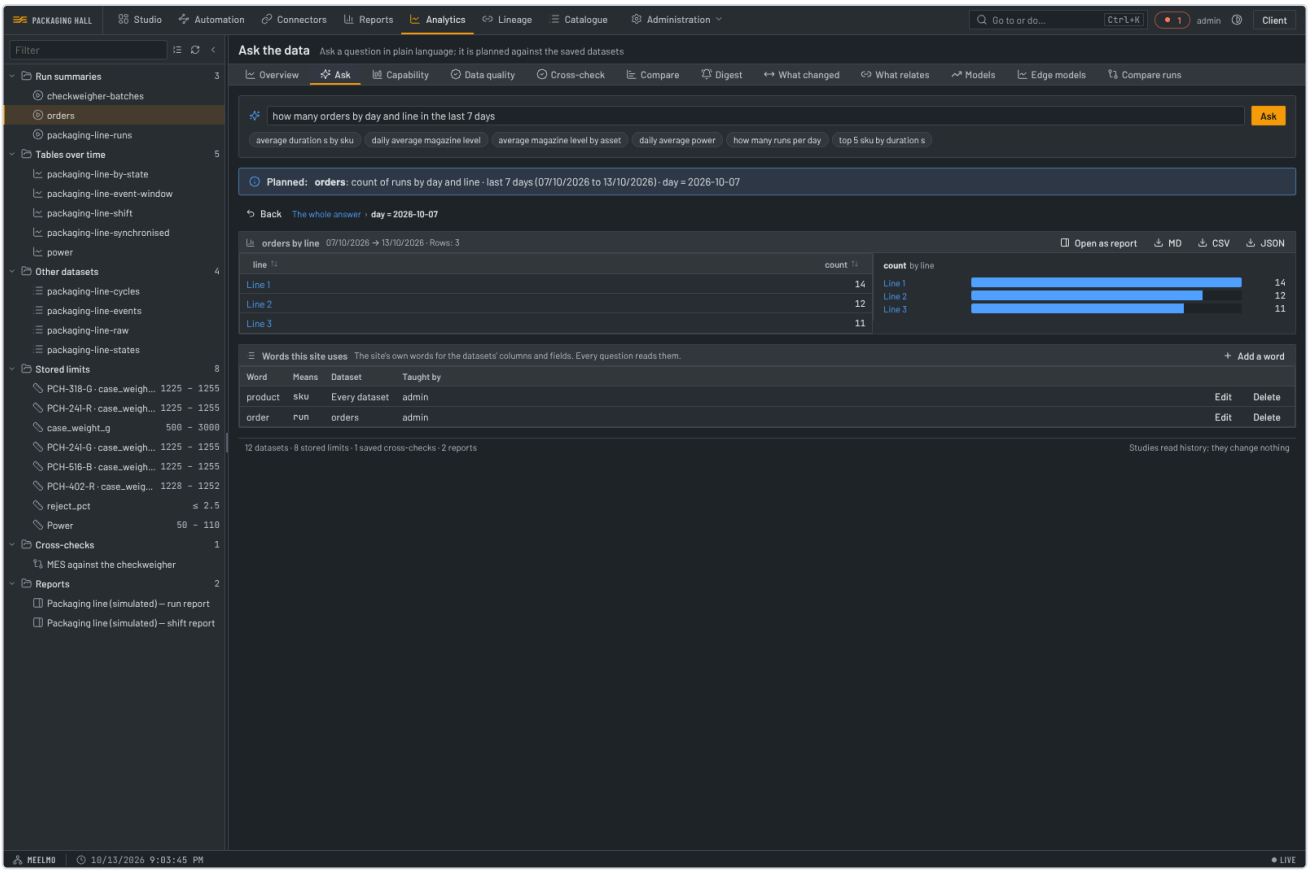
Narrowing to a group. Click a bar, a column, a category's name under the chart, or a group in the table (the underlined value in a grouping column; a double click or Enter on the row does the same) and the question is asked again, narrowed to that group: the group becomes a filter (`sku = PCH-318-G`) and stops

being a grouping, so *average weight by sku and line* narrowed to PCH-318-G is the average by line within PCH-318-G, and narrowing the last grouping leaves one number for the period. A column of a two-grouping chart narrows by both its category and its series.



How many orders by day and line in the last 7 days: the plan, the table of each day and line with its count, and under it a column per day stacked by line — Line 1, Line 2 and Line 3 each in its own colour, with the legend below — because a count adds up to the day's whole.

The **Planned** line ends with what was picked (· *sku = PCH-318-G*). Each step is a new address — `/analytics/ask?q=…&where=sku%3DPCH-318-G`, the same `where` the process study uses — so a narrowed answer can be sent, and the browser's **Back** undoes a step. Above the answer, a trail shows each step: **The whole answer** returns to the question as asked, each earlier step returns to it, and **Back** undoes the last.



The same question narrowed to one day by clicking its column: the Planned line ends with the day picked, the trail above the answer reads Back, The whole answer and the day, and the answer is now the orders of that day by line alone, as a table and ranked bars.

In a ranking of runs (*top 10 runs by weight*) there is nothing to group: clicking a run opens the **Investigate** page at the moment it began, over its dataset's signals, as a cross-check's item does.

Open as report. Somebody who may design reports sees **Open as report** at the top of the answer. It saves the answer as a Report Studio report and opens it in the designer (`/reports/design/…`): the saved dataset as the data source, the plan's filters, grouping, measures, order and row limit as the source's query (run by the same code as the answer), a table of the groups and their measures and, under it, a bar chart with the second grouping as series. The report is named after the dataset (`ask-orders` , `ask-orders-2` when that is taken), titled with the question, and looks back as far as the question's period was long, so running it later covers the same length of time ending then. A report's data source has no hour, day, week or month columns — only the Ask page derives those — so a grouping by one of them, and a filter on a picked day, are not carried over: the report lists the rows its filters keep, with the measured columns totalled the way the question measured them (a count of the rows for *how many*), and draws them over time, and its description says so. Saving needs the Report Studio's **Design** permission; without it the button is not shown.

Who may ask. Asking reads history, so it needs the `ReadHistory` permission and a licence that includes historical queries; a question answered from a run summary also needs `ReadEvents` , as the runs' identifiers are the plant's. Run summaries are answered on a node or a server attached to one. The address carries the question (`/analytics/ask?q=…`), so an answer is a link somebody can open.

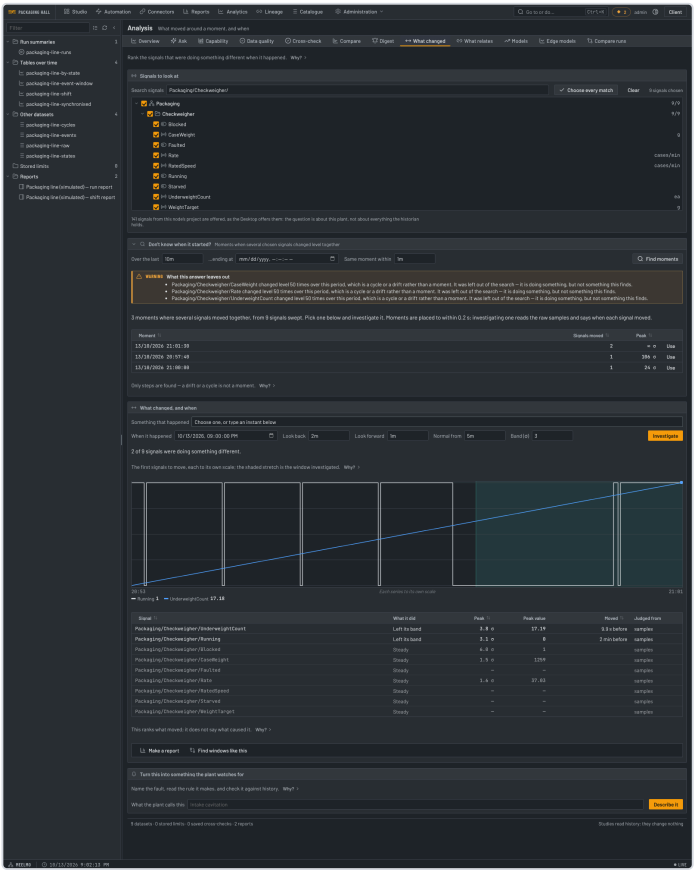
Over the API and to AI clients. `POST /api/v1/analysis/ask` with `{"question": "…", "dataset": "…", "now": "…", "where": ["sku=PCH-318-G"]}` (only the question is required; `where` narrows the answer to groups of it, each a grouping of the question — any other field is refused) returns the outcome (`answered` , `choices` or `refused`), the plan in words (`planned`) and as a structured query (`plan` : filters, grouping, measures, sort, limit and the window — the same query a Report Studio data source runs, executed by the same code), the columns and rows, and the chart suggestion (`bar` , `line` , `column` , or for two groupings

`grouped` or `stacked` with the second grouping as `series`); a refusal over unknown words also names the dataset that came `closest`. AI clients have the same as the MCP tool `analysis_ask`, which reads the site's words too.

The words are `GET /api/v1/analysis/synonyms` (optionally `?dataset=`, which lists that dataset's words and the ones for every dataset), `POST` with `{"word": "product", "means": "sku", "dataset": "orders"}` (leave `dataset` out for every dataset; the same word in the same place is replaced), and `DELETE /api/v1/analysis/synonyms/{id}`. Reading them needs the history permission; teaching and deleting need the datasets' permission (`ManageDatasets`).

What changed

"Rank the signals that were doing something different when it happened." Each chosen signal is compared against its own recent behaviour around a moment, and the ones that departed are ranked. **Why?** beside the sentence unfolds the explanation.



What changed after an investigation: the chosen checkweigher signals, the moment investigated, and the ranking of every signal asked about with what it did, how far it went, when it moved and what it was judged from.

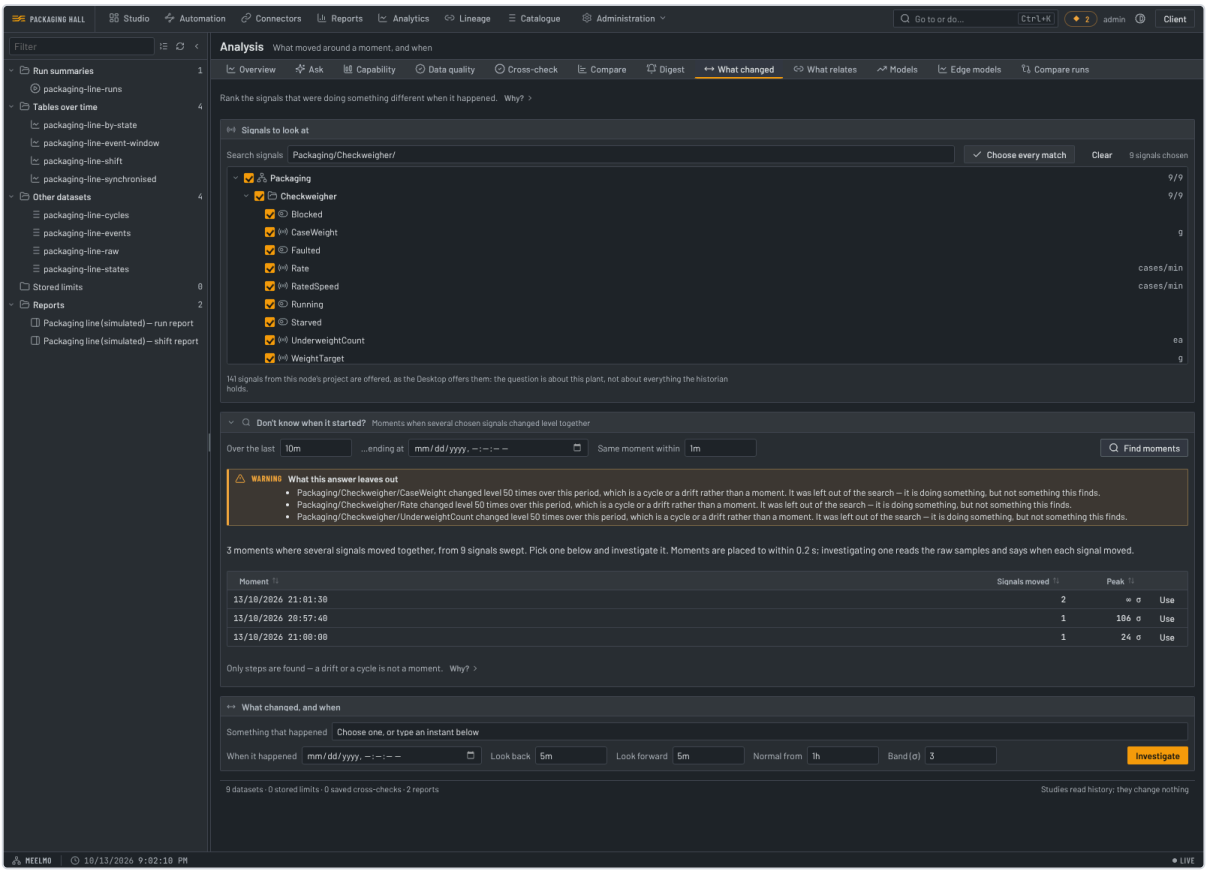
Don't know when it started?

A collapsible panel for when nobody can name the moment: "Moments when several chosen signals changed level together".

Field	Meaning	Demonstration value
Over the last	how far back to sweep (`8h`, `24h`, `7d`)	`24h`
…ending at	the end of the sweep; empty means now	empty
Same moment within	how close in time steps must be to count as one moment	`1m`

Field	Meaning	Demonstration value
Find moments	runs the sweep	

The result is a table **Moments when several signals moved together**: **Moment**, **Signals moved** (how many stepped together), **Peak** (the largest departure, in standard deviations) and **Use**, which sets that moment as the instant to investigate. A line above says how precisely moments are placed ("Moments are placed to within …; investigating one reads the raw samples and says when each signal moved."). Only steps are found — a signal whose level jumped once and stayed. A drift or an even cycle is not a moment, so an empty result ("No step change was found in that period.") means no step was found, not that nothing happened.



Find moments over the last 24 hours on the checkweigher: a table of moments when several chosen signals changed level together, each with how many signals moved and the peak departure, and a Use button to investigate it.

What changed, and when

Field	Meaning	Demonstration value
Something that happened	a list of what the node's own detection rules raised in the last week — when, what the site calls it, and where. Choosing one sets the instant. "Choose one, or type an instant below"	a checkweigher reject event
When it happened	the instant to investigate, if typed by hand	`29/09/2026 05:30`
Look back	how much before the instant is inside the investigated window	`5m`
Look forward	how much after it	`5m`

Field	Meaning	Demonstration value
Normal from	how long before the window each signal's normal behaviour is learned from	`1h`
Band (σ)	how many standard deviations from normal count as "doing something different"	`3`
Investigate	runs the ranking	

When nothing was detected the list says "Nothing was detected in the last week. Type an instant, or find moments above." The page refuses to run without at least one signal ("Choose at least one signal to look at.") or without a moment ("Pick something that happened, or type an instant.").

Reading the ranking

A sentence first — "3 of 7 signals were doing something different." — then a chart, **The signals that moved, around the moment investigated**: the first signals to move, each to its own scale, from the start of normal to the end of the window, with the investigated window shaded. Then the table **Every signal asked about, most involved first**:

Column	Means
Signal	the signal id
What it did	*Step change*, *Left its band*, *Steady*, *No baseline* or *No data*
Peak	the largest departure from its normal, in standard deviations
Peak value	the reading at that peak, in its unit
Moved	when it moved, relative to the instant: "2 min before" is a candidate cause, "30 s after" a consequence
Judged from	*samples*, or *N buckets* when the record was too long to read raw; *part of the period* when even the buckets did not cover the whole window, in which case "steady" has not been shown to be steady

"No signal left its own band" is itself a finding: whatever happened is not visible in the signals chosen, and the next question is which ones are missing. The caveat is printed with every answer: this ranks what moved; it does not say what caused it.

After the investigation

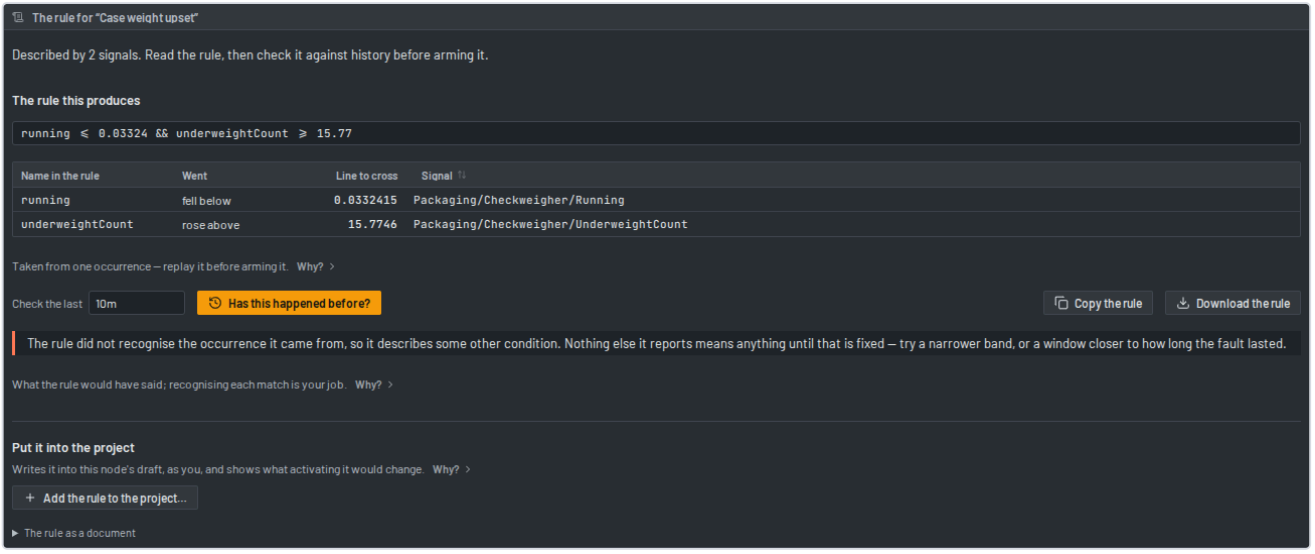
Button	Does
Find windows like this	Searches the sweep's period for other stretches shaped like this one, made of the signals that moved. The table **Windows that look like the one investigated, closest first** lists each with a **Distance** and a **Use** button to investigate it in turn. A distance only ranks: read the order and the gaps, not the number
Make a report	Writes the investigation out as a report and saves it among the node's reports (offered to whoever may design reports). "Saved among this node's reports as “…”." with **Open it**

Turn this into something the plant watches for

"Name the fault, read the rule it makes, and check it against history."

1. **What the plant calls this** — a name, for example `Checkweigher underweight burst`. A rule nobody named is one nobody recognises in a list a year from now.
2. **Describe it** — produces **The rule for "…"**: a table of **The signals the rule waits for**, each with **Name in the rule**, **Went** (*rose above* or *fell below*), **Line to cross** and **Signal**, and **The rule this produces** as text.
3. **Check the last** — a period to replay over, for example `90d` or `6mo` — then **Has this happened before?**. The table **Each time the rule would have spoken** lists **Raised**, **Held** and which occurrence is *this occurrence*. The page then says in words whether this looks like the first time, whether it happened before (look at the dates you recognise), or whether the rule is too loose and describes normal running.
4. Keep it:
    - **Copy the rule** or **Download the rule** hands the text to whoever edits the project's detection rules.
    - **Add the rule to the project…** writes it into the node's configuration draft, as you, and shows what activating that draft would change: whether it **adds** a rule or **replaces** one the plant runs now, every setting that changes (**Now** and **After**), anything else the draft already holds from someone else, and anything the node's validation refuses. **Activate** then makes it a new revision with the sentence you give in **Why, for the revision history**; **Take it back out** removes only this entry. Somebody the node does not let change configuration is told so and keeps the copy and download buttons.

The rule was taken from one occurrence, and the page says so: replay it over history before arming it.



Describing the fault: the rule's signals with the direction and the line each must cross, the rule as text, and the replay over the last months listing each time the rule would have spoken.

Worked example: a burst of underweight cases

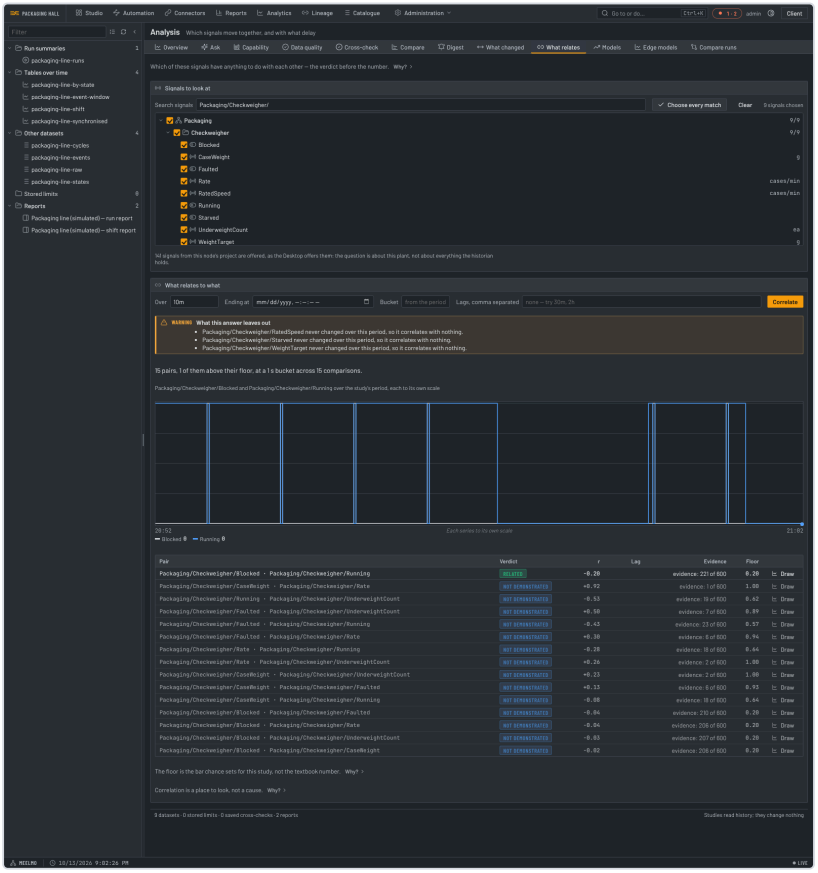
At 05:30 the checkweigher rejected several cases in a row. Nobody is sure what started it.

1. **Analysis** in the top bar. The **What changed** tab is open.
2. In **Search signals** type `Checkweigher/` and click **Choose every match**; then type `BottomSealer/` and **Choose every match** again. The counter shows the signals chosen.
3. Open **Don't know when it started?**, leave **Over the last** at `24h` and **Same moment within** at `1m`, and click **Find moments**.
4. In the table, find the moment near 05:30 with the most **Signals moved** and click **Use**. **When it happened** is filled in.
5. Leave **Look back** and **Look forward** at `5m`, **Normal from** `1h`, **Band (σ)** `3`, and click **Investigate**.

6. Read the ranking top down. A signal that **Moved** *before* the instant (for example `Packaging/BottomSealer/Rate`, "2 min before") is a candidate; one that moved after (`Packaging/Checkweigher/UnderweightCount`) is a consequence.
7. Click **Find windows like this** to see whether the plant has done this before in the swept day.
8. Name it `Checkweigher underweight burst`, click **Describe it**, set **Check the last** to `90d` and click **Has this happened before?**.
9. If the matches are dates the shift recognises, click **Add the rule to the project…**, read what it changes, type a reason and **Activate** — or **Copy the rule** for the engineer who maintains the project.

What relates

"Which of these signals have anything to do with each other — the verdict before the number." There is no target and no threshold: hand it the plant's tags and it says which pairs have anything to do with each other.



What relates over the checkweigher signals: every pair led by its verdict, one pair drawn over the study's period, each series to its own scale.

The **What relates to what** panel:

Field	Meaning	Demonstration value
Over	the period studied	`24h`
Ending at	its end; empty means now	empty
Bucket	the resolution the signals are compared at; empty takes one "from the period"	empty

Field	Meaning	Demonstration value
Lags, comma separated	extra time offsets to try, for a cause that acts later ("none — try 30m, 2h")	`30m, 2h`
Correlate	runs the study	

At least two signals are needed ("Choose at least two signals: a correlation needs both sides.").

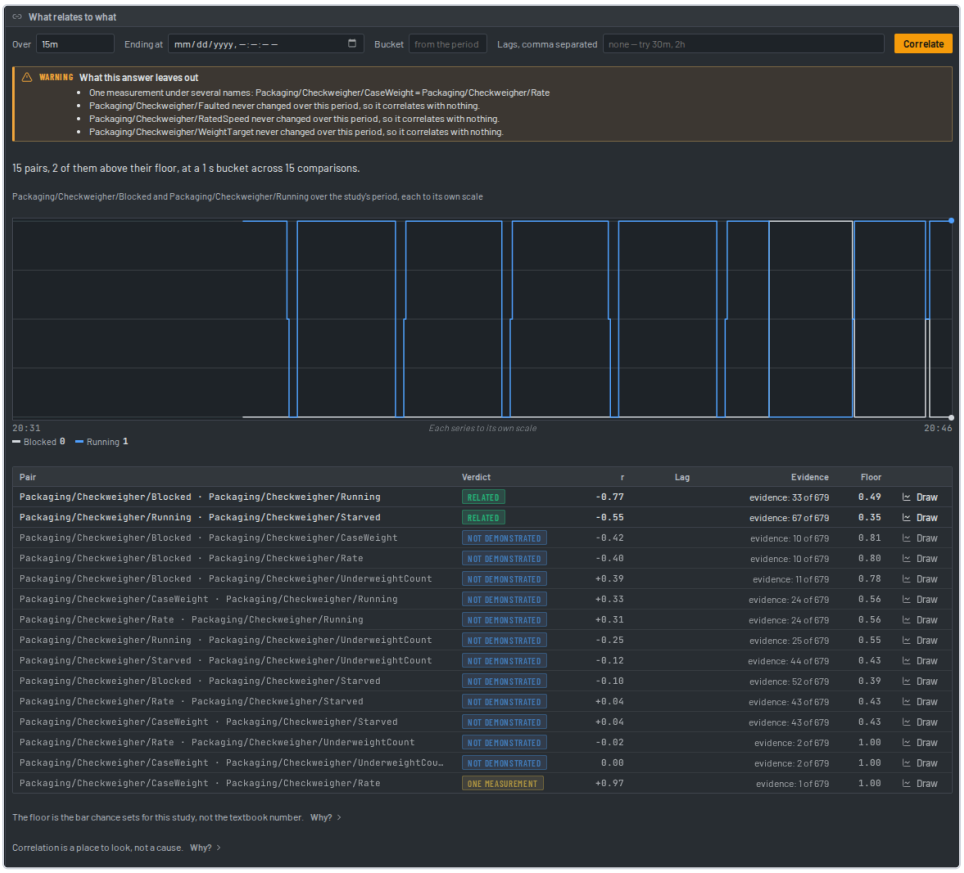
Reading the answer

Above the numbers, **What this answer leaves out** lists what was excluded and why — for example "Packaging/Checkweigher/Blocked never changed over this period, so it correlates with nothing." Then a summary sentence ("10 pairs, 1 of them above their floor, at a 1 s bucket across 10 comparisons.") and, in orange, any groups that are **one measurement under several names**.

The table **Every pair, in the order the study ranks them**:

Column	Means
Pair	the two signals; with a lag, the leading one first (`A → B`)
Verdict	*Related*, *Not demonstrated* or *One measurement* — read this before the number
r	the correlation coefficient
Lag	the offset at which it was found
Evidence	how many independent observations the pair was really worth, for example `evidence: 6 of 601`
Floor	the correlation this pair had to reach before chance stopped being the simplest explanation
Draw	draws the pair above the table over the study's period, each to its own scale

The floor is not the textbook number. A plant record holds far fewer independent observations than samples (a slowly drifting temperature repeats itself), and every pair at every lag is another draw, so the floor is raised for the size of the study. Correlation is a place to look, not a cause: signals sharing a shift pattern or a single upstream setpoint correlate strongly with nothing between them.



A closer look at a correlation answer over ten minutes of checkweigher signals: the signals left out because they never changed, two groups that are one measurement under two names, Running and UnderweightCount drawn together, and every pair with its verdict, coefficient, evidence and floor.

Worked example: does the reject rate follow the sealer?

1. **Analysis ▸ What relates**.
2. Search `Checkweigher/` and **Choose every match**; search `BottomSealer/Rate` and tick it.
3. **Over** `24h`, **Lags, comma separated** `30m, 2h`. Click **Correlate**.
4. Read **What this answer leaves out** first: signals that never changed (`Blocked`, `Faulted`, `RatedSpeed`, `WeightTarget` in the demonstration) are excluded, not unrelated.
5. Skip the pairs whose verdict is *One measurement* — in the demonstration `Running` and `Starved` are one measurement under two names.
6. For the pair with the highest verdict, click **Draw** and look at both lines together before believing the number.

Process capability

Analytics ▸ Process capability (or the **Capability** tab) answers three questions about one value of the process: can it hold its spec, is it behaving the way it usually does, and where did its level move? It is the capability study REPXplore offered, with the same arithmetic.

What an item is. The study reads one numeric column of a saved dataset, one item per row:

- a **run summary** (one row per run) gives one item per run — the column is that run's aggregate, such as `CaseWeight__avg` for each order's average case weight or `Energy__delta` for what each order consumed. These are offered first, and the run's average is chosen for you;
- a **fixed-interval table** gives one item per interval.

A table that keeps every sample is accepted but warned about: consecutive readings are rarely independent, which makes the control limits tight. Event tables, state tables and sequences are not offered.

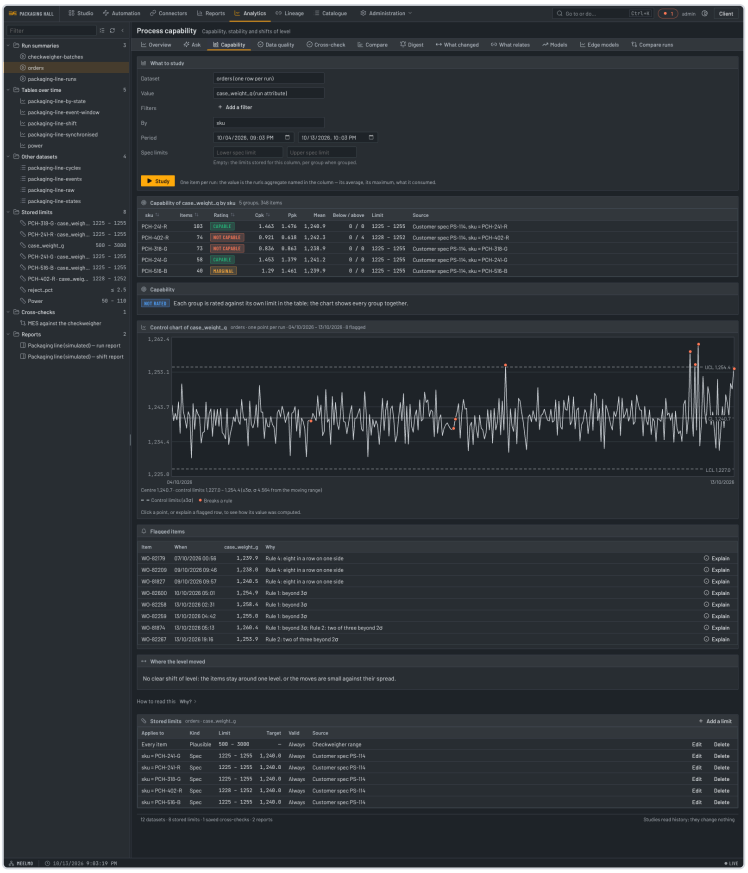
Run attributes are values too. A run summary can join run attributes as context fields — a SKU, a line, or a figure the plant's own database recorded for each order, such as the QA lab's average case weight. Those appear under **Value** marked *(run attribute)* and are read as numbers; rows where the attribute is text are left out and counted in the warnings.

Using it. The form reads top to bottom, one row each:

1. **Dataset** and **Value**.
2. **Filters** (optional, up to three): **Add a filter**, choose a field and type the value — `sku` = `PCH-241-R`, `execution_type` = `WorkOrder`. The fields are a run's `execution_type`, `status` and `asset`, and every context field the dataset joins; add context fields to the dataset to filter by them.
3. **By** (optional): a field to rate each value of separately — each SKU, each line.
4. **Period**: from and to; the latest ninety days by default.
5. **Spec limits**: a lower and/or upper limit. Leave them empty to use the limits stored for this column (see below) — per group when grouped. Without any limit the page still draws the control chart and finds shifts.
6. **Study**.

Reading the answer. Warnings come first — rows without a value that were left out, a dataset that stopped at the server's row limit. Then:

- **Capability by group** (when **By** is chosen): one row per value, largest first — its items, rating, Cpk, Ppk, mean, items outside, the limit it was judged against and where that limit came from. A group with no limit that applies says so. The chart below shows every group together, so it has no capability of its own.
- **Capability**: where the limits came from (typed, or a stored limit's source) beside the title; a rating — *Capable* (Cpk ≥ 1.33), *Marginal* (1.0–1.33), *Not capable*, or *Not rated* when there are fewer than ten items or no limit — and a table of Cpk, Cp, Ppk, Pp, the mean, the short-term and overall spread, the items actually below and above the limits, and the parts per million a normal distribution would put outside. Cp and Cpk use the short-term spread (the moving range of consecutive items); Pp and Ppk the overall spread. Under thirty items the indices are rough, and the notes say so.
- **Control chart**: the items in time order, the centre line (CL), the control limits (UCL/LCL, dashed, ±3σ from the moving range) and the spec limits (USL/LSL, solid, in their own colour). Control limits are what the process does by itself; spec limits are what is required of it. A red dot marks an item that breaks a Western Electric rule — 1: beyond 3σ; 2: two of three beyond 2σ on one side; 3: four of five beyond 1σ on one side; 4: eight in a row on one side of the centre — and a square in the spec lines' colour an item beyond a spec limit. The dates of the first and last item are under the axis, and one line under the chart gives the centre, the control limits and σ. Hover an item to see which run or interval it is. Where the level shifted, each level's mean is drawn as a dotted line. At most 1,000 items are drawn; past that the latest are kept and a note says so.
- **Where the level moved**: each shift with the item it starts at, the level before and after, the change, and its effect size (Hedges' g). A shift is kept only when both sides have at least seven items, the Welch t is at least 4 and the effect at least 0.5.
- **Flagged items**: every item that breaks a rule or a spec limit, as a row — the chart said in words.



Process capability of orders · case_weight_g by sku over the last nine days: the form with the dataset, the value marked run attribute, By sku and the period; Capability by group with a row per SKU, each rated against its own customer spec with its Cpk, Ppk, mean and the items below and above it — PCH-318-G, the widest spread, and PCH-402-R, heavy since yesterday, below the others; the control chart of every order in time order with its centre and control limits and the points that break a rule in red; the flagged orders, each with Explain; and the limits stored for the column.

A point that breaks a rule is a change worth looking into, not necessarily a fault; **How to read this** at the foot of the page repeats the server's own notes, in your language. The same study is available to AI clients as the MCP tool `analysis_process`, with the same filters and grouping.

Over the REST API every analysis answer keeps its English sentences in the fields they have always been in (`notes`, `warnings`, `steps`, `verdict`, `detail`, `message`, `reservation`…). Beside each is a field of the same name ending in `Said` (`notesSaid`, `detailSaid`, `reservationSaid`…) holding the sentence as a catalogue `key` and its `args` — each argument a `text`, `number`, `at` (an instant), `day`, `month`, `seconds`, a catalogue word (`key`), a nested `phrase` or a `list` — plus `countedBy` for a sentence whose form follows a count. A list's entries line up one for one with the English, and an entry is `null` for a sentence with no key. A refusal carries the same beside its `message`: the error body gains `messageSaid` where the server composed the sentence itself, and has no such field otherwise. The MCP tools leave all of these out, errors included: an assistant reads the English.

Stored limits

A spec is usually a property of the product, not of one study: the case weight window for PCH-241-R, a tighter one for PCH-402-R. Stored limits keep it once, per dataset column:

- **Applies to**: conditions as `field = value`, separated by commas — `sku = PCH-241-R`. Empty applies to every item.
- **Kind**: a **Spec** is what the process must hold, and is what a capability study judges against; a **Plausible** limit is what a value can physically be — a duration cannot be negative, a case of pouches does not weigh 0 g — and is what Data quality flags values outside of. A study never uses a plausible limit, and the quality check never uses a spec.

- **Lower**, **Upper**, and an optional **Target**.
- **Valid from**: a revised spec is a new limit with a later date; the old one keeps judging the studies of the time it applied to.
- **Source**: the standard, customer spec or procedure the numbers come from, shown wherever the limit is used.

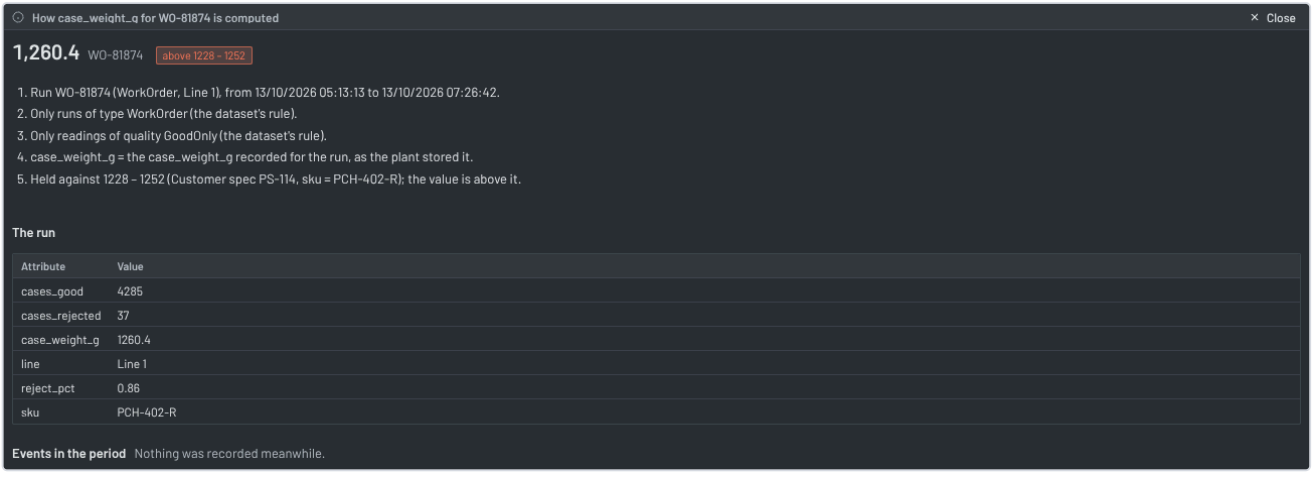
Of the limits whose conditions a study (or a group of it) fixes and whose validity overlaps the period, the most specific applies — a limit for one SKU beats the one for every SKU — and among equally specific ones the newest. A limit for one SKU is not used for a mixture of SKUs: a study that neither filters nor groups by that field says so in its notes.

The limits stored for the dataset and column in hand are listed at the foot of **Process capability**; somebody who may change datasets (who designs reports) can **Add a limit**, **Edit** and **Delete** there, and after a study with typed limits, **Store these limits** keeps them with the study's filters as conditions. On a node the limits are configuration like the datasets: saving one activates a revision, so it can be traced and rolled back. Over the API they are `GET`, `POST` and `DELETE` `/api/v1/datasets/limits`.

How a value was computed

A number on the chart is only useful if somebody can check it. Click any point of the chart, or **Explain** on a row of the flagged table, and a panel opens under the chart saying how that one value was made, the way REPXplore explained a number of a report:

- **the value, large, with its limit beside it**: green *within* the stored limit, red *below* or *above* it;
- **the steps, numbered**: the run (its identifier, type, asset, start and end) or the interval; the dataset's own rules that chose it (run type, asset, attribute conditions, quality); the formula in words, such as *the average of Packaging/Checkweigher/CaseWeight's readings from the run's start to its end*; how many readings lay in that window and how many their source marked bad; and the limit it is held against, with its source and conditions;
- **the readings themselves**, drawn over the run or interval, so the number can be set against the curve it was reduced from;
- **the run's attributes** (SKU, line, whatever the plant records) and the **events** recorded meanwhile;
- **notes** that change how far the number can be trusted: a run that has not ended yet, a window too long to count reading by reading, a signal this account may not read.



How case_weight_g for one order is computed: the value large with its limit beside it, the numbered steps — the run with its type, line, start and end, the dataset's rules that chose it, the value as the MES recorded it for the run, and the SKU's spec it is held against — then the run's attributes, the events recorded meanwhile and a note when the run had not ended.

A value that is a run attribute has no readings to draw: its step says it is the figure the plant stored for the run.

The value is **recomputed, not remembered**: the dataset is built again over exactly that window, as the person asking, so the explanation cannot disagree with the study — and a number the person may no longer read is refused rather than explained. Explaining a run summary's value needs the events permission, like studying it. Over the API it is `POST /api/v1/analysis/explain` with the dataset, the column and the item's instant (`at` , as a study's chart gives it); an assistant connected over MCP uses the `analysis_explain` tool.

Data quality

Analytics ▸ Data quality (or the **Data quality** tab) asks the question to ask before any other: can the numbers in a dataset be believed? It is REPXplore's quality check, over the same saved datasets **Process capability** reads — a run summary gives one item per run, a fixed-interval table one per interval — so a column that looks wrong here is the column a study would have been made of.

Using it. The form reads top to bottom:

1. **Dataset**.
2. **Columns**: every numeric column is ticked; clear the ones you do not want checked. The box in the list's header ticks or clears them all.
3. **Filters** (optional, up to three), as in Process capability.
4. **Period**: the latest ninety days by default, as in Process capability.
5. **Check**.

The checks. Each item of each column is counted by the first check that finds it wrong, in this order, so a column's counts add up and -9999 is a sentinel rather than also an outlier:

Check	What it finds
Missing	A row with no value in the column — an absent value is not a zero — or, for a run attribute, text that is not a number.
Not good	A reading the historian itself marked bad or uncertain. Only for columns whose dataset carries quality codes; otherwise such readings never reach the dataset.
Sentinel	A stand-in for "no reading": -99.9999, -999.9, ±9999, ±99999 or the largest float a driver writes, where it lies outside the range of the column's other values (9999 among readings of 9990–10010 is a reading); the constant the dataset writes for a missing value; 0 where the other values never come near it (zeros at most 10 % of the values, all other values of one sign, the smallest at least a tenth of their median size); and the column's highest or lowest value when it occurs three or more times beyond the outlier threshold — a clamp, not a reading.
Implausible	Outside the column's stored **Plausible** limit (see Stored limits), found for each item's own field values and date, so a limit for one SKU judges that SKU's items.
Outlier	More than 5 robust deviations from the median — the median absolute deviation × 1.4826 — when at least 20 values are left to judge by.
Stuck	The same value, exactly, for 10 or more items in a row: a frozen transmitter, a value copied forward.

For the table's rows themselves, REPXplore's checks of a table filling up:

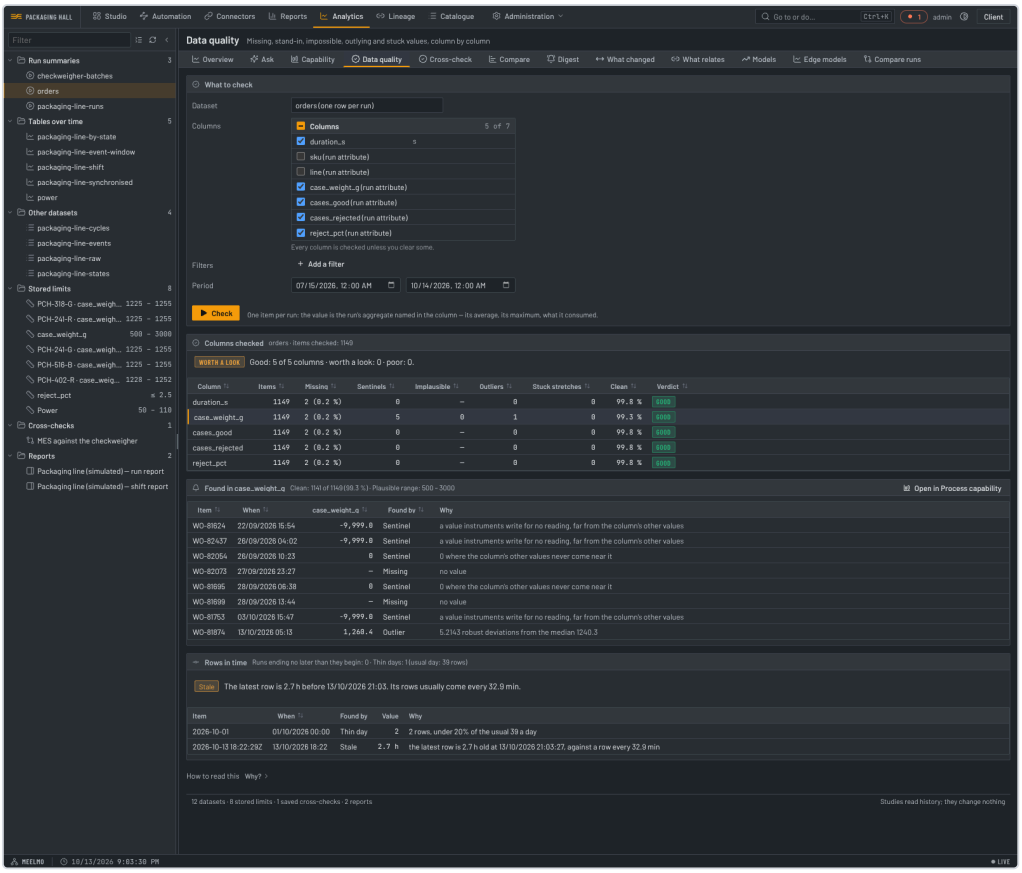
Check	What it finds
Freshness	How long before the end of the period — or now, when the period runs past it — the latest row is, against the table's cadence: a fixed-interval table's own interval, or else the usual (median) spacing of its rows, such as a run summary's runs. The latest row is **stale** when it is older than 3 cadences and older than 5 minutes (a one-second table is a few minutes behind as a matter of course while the historian's buffer drains). With fewer than three rows there is no spacing to judge by, and only the age is given. A period with no end is not judged.

Check	What it finds
Gaps	In a **fixed-interval table** over a calendar period: intervals with no row between the first row and the last, and at the edges of the period — every interval from its start to the first row, and every whole interval from the last row to its end (or now); the interval still under way is not missing.
Thin days	Days with rows, but fewer than a fifth of the usual (median) day's. Days are UTC days, and only the ones the period covers whole are judged, when there are at least seven of them: the first and last day of a period, and today, are only partly in it. The last day is mentioned under **How to read this** when it is thin, since it may still be filling. A day with no rows at all is not thin; in a fixed-interval table it is a gap.
Durations	In a **run summary**, runs whose end is not after their start (a run still going is not one of them).

Gaps, freshness and thin days are counted over every row of the table, whatever the filters: a filter leaves rows out, it does not make a table late or a day thin.

Reading the answer. Warnings come first, as on every analysis page. Then:

- **Columns checked**: the overall verdict — *Good*, *Worth a look*, *Poor* or *No data*, with how many columns are each and the weakest — and a table with one row per column: its items, missing (count and share), not good (shown only when any column has some), sentinels, implausible (— when no plausible limit is stored), outliers, stuck stretches, the share of clean items, and its verdict. A column is *Good* at 98 % clean or more, *Worth a look* from 90 %, *Poor* below. A table with gaps (inside or at the edges), bad runs, thin days or a stale latest row is at best *Worth a look* overall.
- **Found in** the chosen column — the weakest when the check has just run; choose another row to switch. Every item found wrong, in time order: which run or interval, when, its value, the check that found it and, in the server's words, why. At most 500 are listed per column; the counts in the table are always complete. **Open in Process capability** opens the same dataset and column there, with the same filters and over the period the check was run for.
- **Rows in time**: first the freshness — **Fresh**, **Stale** or *Too few rows to judge* — with the latest row's age before the end of the period (or now) and the cadence it was judged against. Its heading line counts the missing intervals and gaps, the intervals missing before the first row and after the last, the runs that end no later than they begin, and the thin days with the usual day's rows. Below, each finding as a row, in time order: a gap (including the ones at the edges), a thin day with its rows, the stale latest row, a bad run.



Data quality of the orders dataset with the SKU and line cleared, since they are text: Columns checked says Worth a look overall, every column Good on its own, case_weight_g with its missing values, sentinels and outliers counted; Found in case_weight_g lists each order found wrong with its value, the check that found it and why; Rows in time says the latest order started hours before the end of the period against a row every half hour, and lists the maintenance day as a thin day.

How to read this at the foot repeats the server's own rules for each check. The same check is available over the API as `POST /api/v1/analysis/quality` and to AI clients as the MCP tool `analysis_quality`, where the outlier threshold (`outlierZ`) and the length of a stuck stretch (`stuckRun`) can also be given.

Worked example: before trusting a capability. A study of `orders` · `case_weight_g` over the last month rates every SKU *Not capable*, and its control chart falls to the bottom of the plot twice. **Data quality** on the same dataset shows the column's sentinels: orders that read -9999 and 0 (the QA scale did not report, and the MES wrote what it had) where the rest are 1,225–1,255 g. So the *Not capable* was about the data, not the filling. Storing a **Plausible** limit of 500–3,000 g for the column on the Process capability page means the next check also catches a reading that is wrong without being an obvious stand-in — 250 g from a scale weighing one pouch instead of a case. Studying the nine days since, which hold no sentinel, rates the SKUs on what the line did.

Cross-check

Analytics ▸ Cross-check (or the **Cross-check** tab) holds two sources that should agree against each other: which items only one of them has, and, for the items both have, whether what each says about them is the same. It is REPXplore's cross-check of two databases of one plant, made of saved datasets.

The commonest pair. Meelmo has two sources of runs: the ones its detection rules find in the signals, and the ones a run source reads from the plant's own production database (see *Runs from a production database: run sources* in *the semantic model chapter*). Both land in the semantic model, so a run summary reads both. An order the MES recorded and Meelmo never saw — or saw with a different weight

— is exactly what this page finds. Two different datasets can be the two sides, and so can one dataset filtered two ways: the left side `execution_type=order`, the right side `execution_type=WorkOrder`. So can two run sources of one database, as in the example below.

The definition.

- **Left side** and **Right side**: for each, a **Dataset** (a run summary or a table over time), the **Item key** each item is matched by — a run's `identifier`, its `execution_id`, a context field such as an order number the plant recorded, or *when each began* — an optional **Key pattern**, and optional **Filters** written `field=value`, separated by semicolons.
- **Matching the items.** By name, the **key rules** make keys written differently comparable: ignore upper and lower case, ignore leading zeros (024518 = 24518), keep only letters and digits (24-A518 = 24A518). A key pattern is a regular expression whose group in brackets is the shared key: `^(\d{6})-\d+$` takes order 245180 out of pallet label 245180-03. By time, the **Time tolerance** says how many seconds apart two items may begin and still be one; the closest pairs are matched first and each item is used once, so two runs a few seconds apart cannot both claim the one record between them. Choosing time on one side chooses it on both.
- **Right side later by**: the right side's period runs this many days longer, for items that reach it later.
- **Period**: the latest month by default, replacing both datasets' own windows.
- **Values to compare**: pairs of columns, one from each side, that should hold the same number — a run's average weight and the weight the plant recorded — with a **Tolerance** as an amount or, ticking **as %**, as a percentage of the left value. A pair without a tolerance is compared and not judged. Up to eight pairs; none checks only which items are missing.

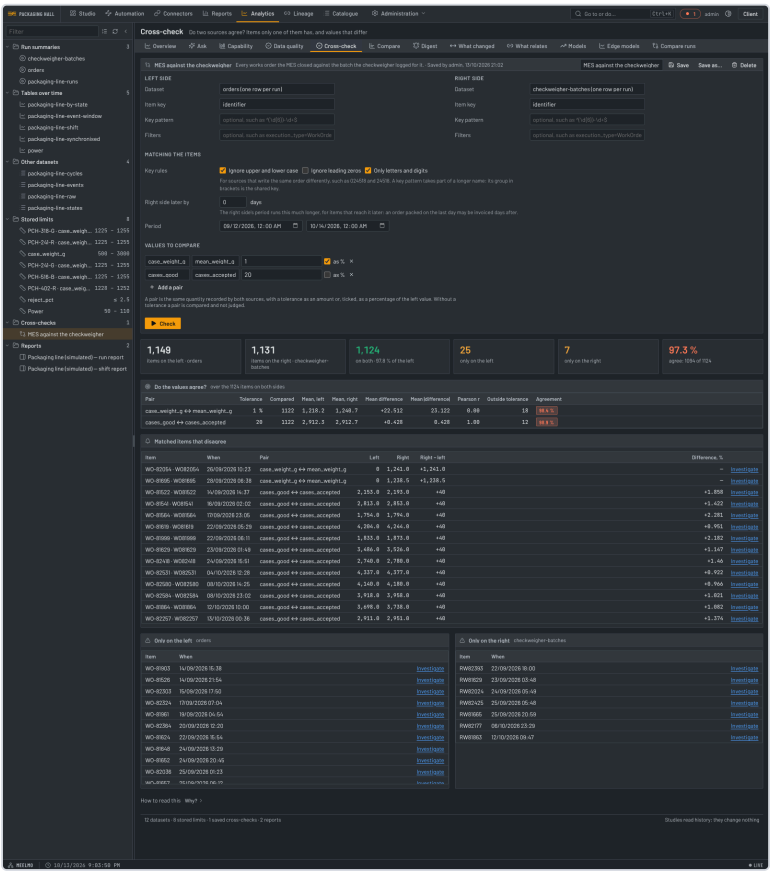
Reading the answer. Warnings first, then six counts in a row: the items on each side, the items on both (and what share of the left that is), the items only on the left, only on the right, and the **agreement rate** — the share of compared items on which every judged pair agrees. An order whose case weight agrees and whose case count does not counts as disagreeing; each pair's own rate is in its row. Then:

- **Do the values agree?**: per pair, the tolerance, how many items were compared, the mean on each side, the mean difference (right − left, so a side that reads systematically higher shows here) and the mean absolute difference, Pearson's r, how many are outside the tolerance, and the pair's agreement.
- **Matched items that disagree**: each value outside its tolerance, largest first, with both values, the difference and the difference in percent of the left value. At most 200 are listed; the title says when there are more.
- **Only on the left** and **Only on the right**: the items one side has and the other does not, with when each began.

Every item in these tables ends in **Investigate**, which opens **What changed** at the instant the item began with its dataset's signals already chosen (the compared column's first, at most six) — what the plant recorded while the order was packed, which is the next question about an order one source missed or the two sources disagree on. No page shows one run by its address, so this is where an item leads; a dataset made only of run attributes, with no signals, opens its **Process capability** study instead.

Saving a cross-check. A check worth running again is saved under a name, as REPXplore saved them. **Save** asks for a name and a description the first time, then saves over the open check; **Save as…** keeps a copy under a new name; **Delete** removes the open check (the datasets it reads are not changed). What is saved is the definition — both sides, the key rules, the pairs with their tolerances, later-by and the time tolerance — and not the period, since a saved check is run again over new periods; who saved it and when is kept. The list above the form opens a saved check and runs it, and so do the explorer's **Cross-checks** folder and a link: `/analytics/crosscheck?name=MES%20against%20the%20checkweigher`. Names are unique, at most 80 characters, and a site keeps at most 100 checks. Saving and deleting need the datasets' permission (somebody who designs reports); anybody who may read history can open and run one. On a node saved checks are configuration beside the datasets, revised and rolled back with them.

Example: the MES against the checkweigher. The packaging hall's MES closes each works order with the QA lab's average case weight and the cases made; the checkweigher logs a batch of its own for every order it weighs, with its mean weight and the cases it accepted, and writes the order number without its dash (`WO82222` for `WO-82222`). The saved check **MES against the checkweigher** matches `orders` to `checkweigher-batches` by `identifier` with *Only letters and digits* ticked, and compares `case_weight_g` with `mean_weight_g` (1 %) and `cases_good` with `cases_accepted` (20 cases).



The saved cross-check MES against the checkweigher, run over the last month: both sides by identifier with Ignore upper and lower case and Only letters and digits ticked, the two pairs with their tolerances; the counts — the orders on the left, the batches on the right, those on both, the orders the checkweigher never logged, the batches no order explains and the agreement rate; Do the values agree? per pair with the mean on each side, Pearson's r and how many are outside the tolerance; the matched orders that disagree, among them the orders whose QA weight read 0 and those whose case counts differ by forty; and the items only on the left and only on the right, each with Investigate.

The disagreements are the orders whose QA weight was a stand-in (0 or -9999), the orders for which the checkweigher accepted forty cases more than the MES counted, and a day on which Line 2's checkweigher read 1.6 % heavy; only on the right are the rework batches (`RW…`) nobody ordered; only on the left, the orders packed while the checkweigher was bypassed.

What to look out for. Items near the start and end of the period may be on one side only because one source recorded them just before it or just after it. When two items of one side get the same key once made comparable, only the first is used and the notes say how many collided, rather than adding two items' values together. When no key matches, the notes show what each side's keys look like, so the rules or the pattern can be fixed.

Over the API it is `POST /api/v1/analysis/crosscheck` , with `left` and `right` (each a `dataset` , a `key` , an optional `keyPattern` and `filters`), `pairs` , and optionally `start` / `end` , `rightLaterByDays` , `timeToleranceSeconds` and the key rules; or `name` , the name of a saved check, instead of the definition, with `start` / `end` still choosing the window. It needs the history permission, and the events permission when a side is a run summary. Saved checks are `GET /api/v1/analysis/crosschecks` , `POST` with the definition and a `name` (an empty `id` makes a new one) and `DELETE /api/v1/analysis/crosschecks/{id}` ;

writing needs `ManageDatasets`. An assistant connected over MCP uses the `analysis_crosscheck` tool, which also takes `name`, and finds the saved checks — each with its two datasets, keys and pairs — with `analysis_crosschecks`.

Compare places

Analytics ▸ Compare places (or the **Compare** tab) puts one value of the process side by side — across the places within this plant, or across plants: for each, how many items, their mean and overall spread σ, the Cpk against its own limit with its rating, and a small control chart.

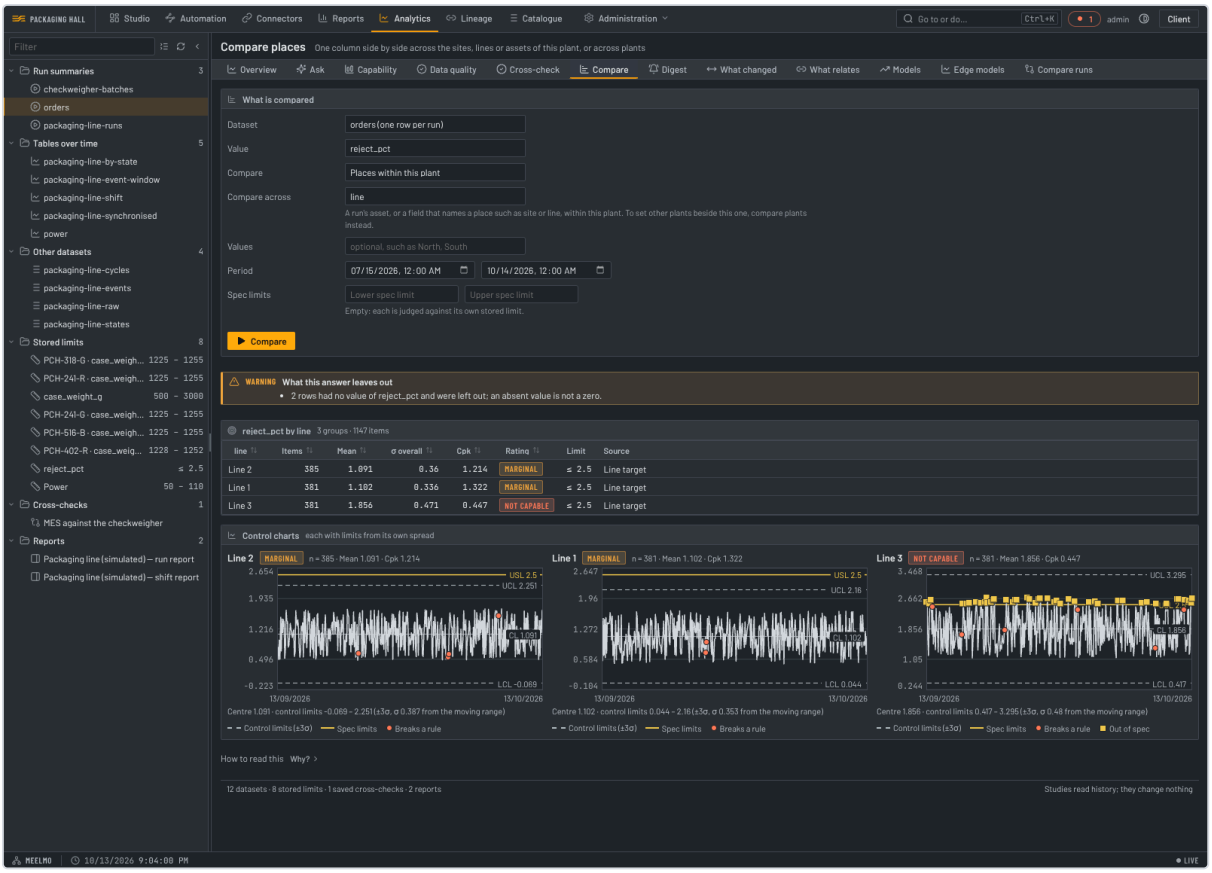
Compare chooses which:

- **Places within this plant** — the places this node's semantic model tells apart: a run's **asset**, or a context field a run source tags each run with, such as **site**, **line** or **unit**. One dataset, split by that field.
- **Plants** — this plant beside the other plants this host reads through **linked nodes** (Administration ▸ Linked nodes; see *Linked nodes*). This is how REPXplore's "compare plants" works in Meelmo: a Meelmo node is one plant, and a web host that has linked other nodes reads them under a read-only account each of them issued.

Places. Dataset and **Value** as in Process capability; **Compare across**, the field — fields whose names say a place are offered first and the first is chosen for you; **Values** (optional), the places to show, separated by commas — every one otherwise, the twelve with the most items; **Period**; and **Spec limits** — leave them empty to judge each place against the limit stored for it (a stored limit with the condition `site = North`, say), or type one pair to hold every place to the same. **Compare**.

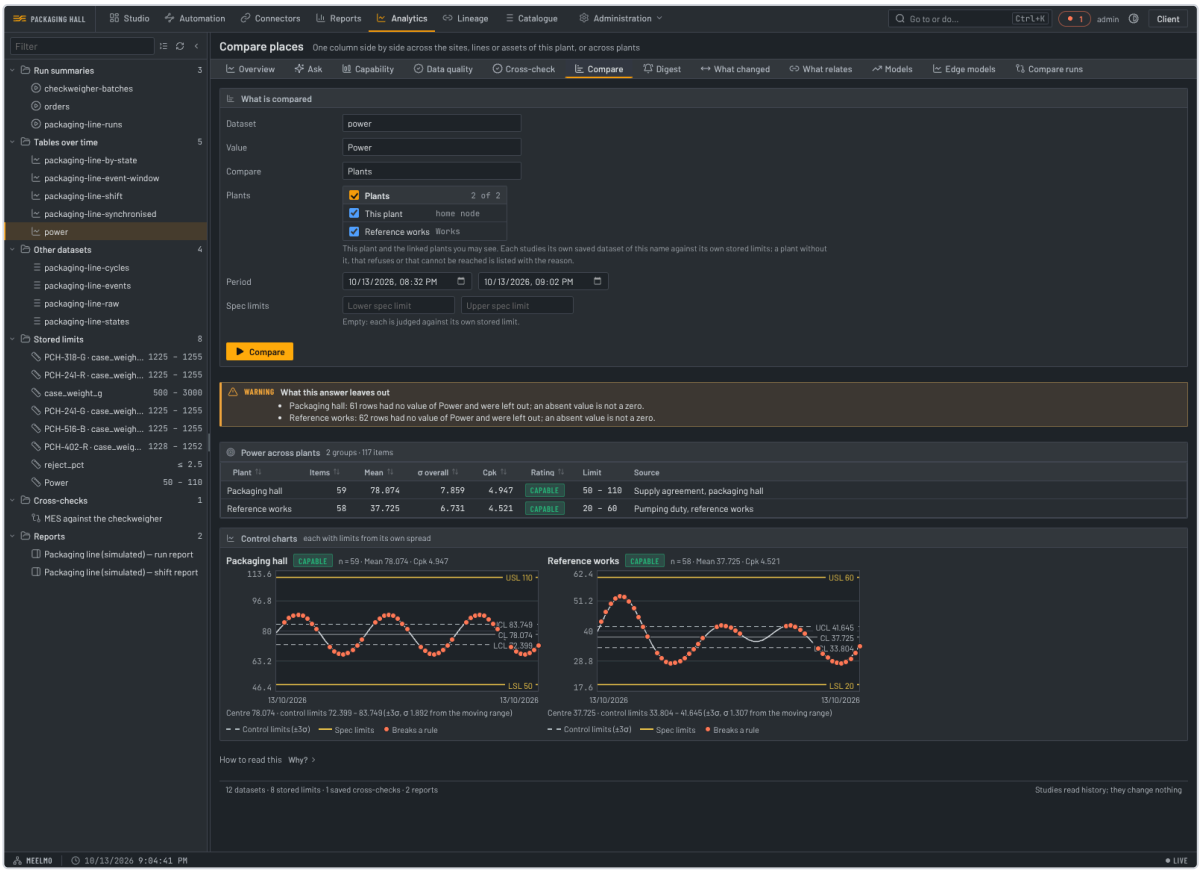
Plants. Dataset and **Value**; **Plants**, a checklist of this plant and each linked plant you may see, all ticked; **Period** and **Spec limits** as for places. A dataset is saved on its own node, so each plant studies **its own saved dataset of that name** — a linked plant's is read through that plant's own API, with the account it issued to this host — and is judged against **the limits stored on that plant**. Give the dataset the same name and column on each plant you want to compare; what it reads there is that plant's business. Typing limits holds every plant to them instead.

Which plants you see is decided here, at the home node, as for every linked read: you need the catalogue permission, and a person whose signal scope does not reach a linked plant's signals is not shown that plant. The linked plant then applies its own rules to this host's account.



Compare places over the orders dataset: reject_pct compared across line, each line judged against the one stored limit of 2.5 %; the table rates Line 3 not capable against Lines 1 and 2, and a control chart of each line at one size, Line 3's with the orders beyond the spec marked.

Reading the answer. A table with one row per place (largest first) or per plant (this one first, then the linked ones in the order they were linked) — its items, mean, σ overall, Cpk, rating, the limit and where that limit came from, or a note when none applies — and then a control chart of each at one size, so their shapes can be compared at a glance. Each chart's control limits come from its own moving range and its spec lines from its own limit, so a noisier line does not widen a steady one's limits. A place with a single item has a mean and no chart; one with fewer than ten items is not rated. A field that has one value in the period says there is nothing to compare it with.



Compare plants over the dataset power, which each plant saved under that name: the packaging hall's line power and the reference works' pumping power over the last half hour, each judged against the limit stored on its own node — 50–110 kW for the packaging hall, 20–60 kW for the reference works — with a control chart of each.

A plant that could not be studied is still a row, with the reason where its rating would be: **Dataset or column missing** (it has no saved dataset of that name, or no such column — save one there), **Refused** (its administrator's grant to this host does not allow the read), or **Unreachable** (it is down or did not answer in time). Its part of the comparison is missing, not empty. When the plants give the column in different units the answer says so, because their figures are then not comparable.

Over the API it is `POST /api/v1/analysis/compare` with `dataset`, `column`, and either `by` (places, the default) or `across: "plants"`; optionally `groups` (the values, or the plants by name, with `home` for this one), `start` / `end`, `filters` and `lower` / `upper`. Across plants each row carries `node` (null for this plant), `state` (`studied`, `missing`, `refused` or `unreachable`) and `reason`. An assistant connected over MCP uses the `analysis_compare` tool, which takes `across` too.

Alarm performance

Alarms (`/analytics/alarms`) is the alarm-performance review: how the alarm system performed over the **last 24 hours**, **7 days** or **30 days**, for all areas or one, measured against ISA-18.2 and EEMUA 191. It opens on the last day. The line under **Analyse** says how many alarms were raised, how many the operator was given (shelved alarms are not) and how many distinct alarms that was; the badges say the overall verdict and how many findings each verdict holds.

- **Findings, worst first** — each with its verdict, one sentence and a suggested action: the average and peak rates, floods and the alarm that started each, the top contributors, chattering alarms with the clear delay that would hold them, fleeting alarms with the raise delay that would stop them, stale alarms, shelved and out-of-service alarms, the priority mix, the times to acknowledge and to return to normal, duplicate and consequential pairs, and each area's own load.
- **Key figures** — each with its value, verdict and the guidance it was held to.

- **Top contributors** — the ten alarms carrying most of the load, with their area and share.

The benchmarks, verdicts and limits are the Desktop's, described under *Alarms*, *Alarm performance*: the server computes the review (`GET /alarms/performance`), and the page, the Desktop window, an assistant's `alarms_performance` and a report's alarm-performance source all say the same thing about the same period. An account scoped to part of the plant is reviewed on that part's alarms. The sentences are said in your language from the figures the server measured.

Process digest

Analytics ▸ Process digest (or the **Digest** tab) answers the morning question: what did the process do since the last period that somebody should hear about? It is REPXplore's daily alerts, over Meelmo's stored limits, and it judges **every stored limit** at once — so it says something only about columns somebody has stored a spec (or a plausible range) for. With no limits stored the page says so and how to store one.

The period and the one before it. Choose **Period** — *The last 24 hours* (the default, run as the page opens), *The last 8 hours* (a shift), *The last 7 days*, or *A chosen period* with its own from and to — and optionally **Limits of** one dataset instead of *Every dataset*. **Run the digest**. Each period is judged against the one of the same length just before it; the line beside the button says both.

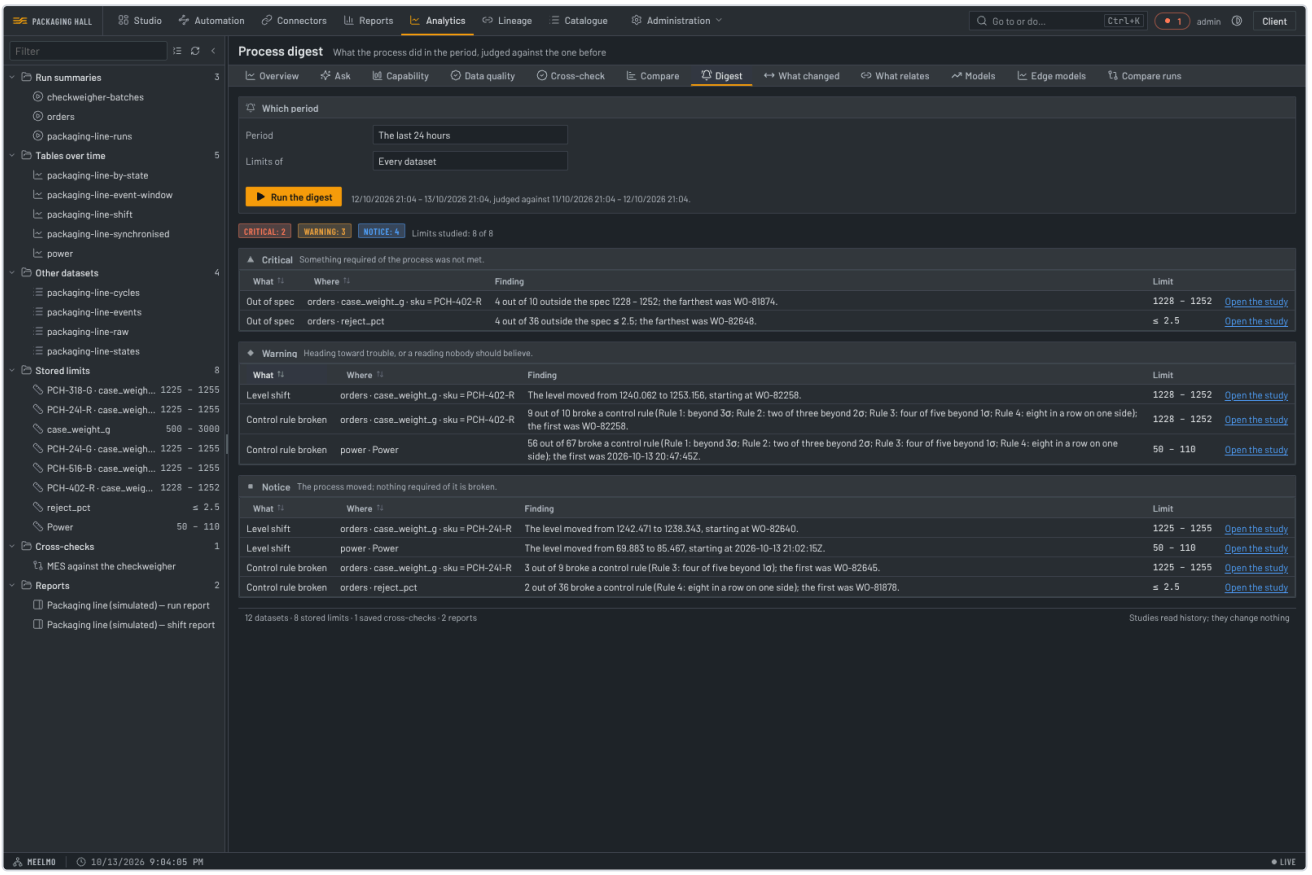
What it looks for, for each limit in force over the period (the newest revision of each, as a study chooses it; a limit's conditions such as `sku = PCH-402-R` select the items it judges):

Finding	Severity	When
Out of spec	Critical	items of the period outside the spec. The finding names how many, the farthest, and the previous period's share
Capability fell	Critical when it fell to *Not capable*, otherwise Warning	the capability rating dropped a band — *Capable* to *Marginal*, *Marginal* to *Not capable*. Only between two periods that could be rated (ten items or more each)
Implausible reading	Warning	readings outside a *plausible* limit: more likely a fault of the measurement than of the process
Level shift	Warning when the new level is nearer the spec, Notice when it moved away	a shift of level (the same test as the study's: seven items each side, Welch t ≥ 4, effect ≥ 0.5) whose new level began in the period — a shift last week is last week's news
Control rule broken	Warning for a point beyond 3σ, Notice for the patterns	items of the period breaking a Western Electric rule **against the previous period's centre and spread**, so today is judged by what yesterday did. With fewer than ten items in the previous period the period is judged by its own centre and spread, and the finding says so

Plausible limits contribute only their readings outside; a column's shifts, rules and capability are its spec's business, so nothing is said twice.

Reading the answer. A line counts the findings per severity and how many limits were studied. Then one panel per severity, **most severe first**: what kind of finding it is, where (dataset, column and the limit's conditions), the finding in a sentence, the limit, and **Open the study**, which opens **Process capability** on that dataset and column filtered to the limit's conditions, **over the period and the one before it** — the two periods the finding compared, so a shift shows the level it moved from and items out of spec show against the ones that were not. **Not judged** at the foot lists the limits that could not be studied and why — a dataset that no longer exists, a column renamed, or a run summary the account

may not read (it needs the events permission). A limit that could not be judged is not a limit that held. A limit with no items in the period is counted in the notes: nothing was made there, which is worth knowing on its own.



The process digest of the last 24 hours against the 24 before: the counts per severity, then the critical findings — PCH-402-R's case weight out of spec, with the farthest order named — and the warnings, a level shift and the control rules the power and the reject share broke; each with the dataset, column and conditions it is about, the finding in a sentence, the limit, and Open the study.

Since yesterday. The **Analytics** hub shows the three most severe findings of the last 24 hours under **Since yesterday**, each a link to its study, and **Open the digest** (with the total when there are more). It says *Nothing needs a look since yesterday* on a quiet day, and nothing at all when no limit is stored.

Delivered by itself — the alert. The digest is also a data source in Report Studio: **Add process digest** on the **Data** tab (see Data sources and parameters). Its rows are the findings over the report's own window, in the columns *Severity*, *Level* (3 critical, 2 warning, 1 notice), *Kind*, *Dataset*, *Column*, *Where*, *Limit*, *Finding*, *Items*, *Count* and *Link*; put a table on the page bound to it and schedule the report as any other (**Report settings ▸ Schedule**). **Send when** on the source turns the schedule into an alert: *Always*, *Anything was found*, *A warning or worse was found* or *Something critical was found*. When the digest found nothing that severe, the run is still written to the report directory — the archive has the quiet days, and the firing counts as done — but it is not mailed or posted, and the node's log says why. A report scheduled at 06:00 over a 24-hour window with **Send when** *Something critical was found* is REPXplore's "sent only when something is found".

Over the API the digest is `POST /api/v1/analysis/digest` with `start`, `end`, optionally `previousStart` and `previousEnd`, and `dataset`; an assistant connected over MCP uses the `analysis_digest` tool. Each finding carries an English `summary` sentence and a `link` — the study's address with `from` and `to` spanning both periods. It is read as the person asking, dataset by dataset, and licensed and permitted as a process study is.

Worked example: a morning alert for the packing hall

A quality lead wants a mail only on the mornings something broke a spec.

1. On **Process capability**, study `orders` / `case_weight_g` grouped by `sku`, and store each SKU's spec (**Store these limits**).
2. Open **Process digest**: the last 24 hours list what the limits found today, most severe first.
3. In **Report Studio**, start a report, **Add process digest** on **Data**, and set **Send when** to *Something critical was found*. Drag the source onto the page: a table of its findings. Keep *Severity*, *Finding* and *Link*.
4. **Report settings ▸ Schedule**: scheduled, at 06:00, *PDF*, **Window** 24 hours, **Mail it to** the lead. Save.

Every morning the report is written to the node's report directory; on the mornings an order left its spec it also arrives by mail, the critical findings first.

Links in a delivered report. Each finding's *Link* opens its study in the browser client. In a PDF, a workbook or an HTML file the link has to be a whole address, so the report writes it against the installation's **public address** — `network.publicAddress` in `web-config.json`, or `MEELMO_WEB_PUBLIC_ADDRESS` (see *Configuration and deployment*). A cell that is an address is a real link in all three: a link in the PDF (it still opens the whole address when the printed text is shortened to fit its column), a hyperlink in the workbook, and a link in the HTML. With no public address set, the links stay paths such as `/analytics/process?dataset=orders&…`, which open nothing from a file, and a PDF or workbook says so under the table.

A process study opened from any of these links — or at an address of your own — takes `dataset`, `column`, `where` (one per condition, `field=value`) and `from` and `to` as ISO-8601 instants, for example `/analytics/process?dataset=orders&column=case_weight_g&where=sku%3DPCH-402-R&from=2026-09-28T06:00:00Z&to=2026-09-30T06:00:00Z`. A period with only one end, one that ends before it starts, or a value that is not a date is ignored, and the page opens on its usual ninety days.

Analytics as your start page

Opening the client at its root address normally shows the operator screens. For somebody who does not operate the plant from this client — a process engineer, a quality lead — that is a screen to walk through on every sign-in.

- On **Analytics**, click **Open here when I sign in**. From then on the root address opens **Analytics**, on this browser and any other: the choice is kept with your profile on the server. The page then says **Your start page**; **Open the operator screens instead** undoes it.
- An administrator can set it for somebody else under **Administration ▸ Users & permissions**: open the person's permissions and choose a **Start page** — *Operator screens*, *Analytics*, *Reports* or *Studio*. It takes effect at their next sign-in.
- A start page the person may not open (for example *Reports* for somebody without `Reports.View`) is ignored, and the operator screens open as before.
- A paired operator station never has a start page: it always shows its pages, whoever is in front of it.
- Somebody with no start page, for whom the operator screens would only say there is nothing to show, and who may read history, is taken to **Analytics** instead. Administrators and authors are not: that screen carries the next step for them (switch the HMI on, make or assign a project).
- The operator screens remain one click away: **Client** in the top bar, and the address `/hmi`, always open them, whatever the start page. The installed app also offers **Analytics** as a shortcut beside **Alarms** and **Studio**.

Compare runs

Analytics ▸ Compare runs (or the tab of that name) opens **Run comparison** — the browser's answer to "why was this order different?". It compares chosen runs against reference runs that define what good looks like.



Run comparison over the last week: the runs found with reference and compare ticks, the signals chosen, and each signal drawn as a band of the reference runs with the compared run over it, followed by the deviation table.

Which runs

Field	Meaning	Default
From, **To**	the window to look for runs in	the last seven days to tomorrow
Align on	*Phase (fraction of the run)* — runs of different lengths are stretched to one axis from start to end — or *Elapsed time* since each run started	*Phase*
Band width (σ)	how wide the reference band is, in standard deviations (0.5 to 10)	`3`
Find runs	lists the runs recorded in the window	

"No run was recorded in this period." means the node detected none. When more than 500 runs are in the window only the oldest are listed, and a warning says to narrow the window to reach the recent ones.

Runs and signals

The **Runs** panel ("References: N · to compare: M") lists each run with two check boxes — **Reference** and **Compare** — and **Run**, **Started** and **Duration**. **Why?** explains: tick the runs that define what good looks like, and the runs to judge against them. Several references give a band; one reference gives only a line

("One reference run: the line is what it did, not a range.").

The **Signals** panel beside it is the same signal choice as the analysis pages. **Compare** runs the comparison.

Reading the comparison

Warnings come first, above the charts, deliberately: every warning means the answer is smaller than it looks — a run left out because it was still running, a signal with no data, a truncated read.

Then one chart per signal: the reference band, the reference mean, and each compared run drawn over it on the chosen axis. Last, the **Deviations** table:

Column	Means
Run	the run's own name
Signal	the signal compared
Worst	the largest absolute deviation from the reference mean
At	where in the run it happened (phase or elapsed time)
RMS	the root-mean-square deviation over the run
Mean	the signed mean deviation — consistently high or low
Outside band	points outside the band, of points compared (`37 / 200`)
Coverage	how much of the run could be compared; below 50 % it is highlighted, because a small deviation over little coverage is not a small deviation

Run comparison is not available while the client is replaying history: it would compare runs that had not happened yet.

Worked example: yesterday's slow order

1. **Compare runs**. Set **From** to a week ago and **To** to today; leave **Align on** at *Phase (fraction of the run)* and **Band width (σ)** at `3`. Click **Find runs**.
2. In **Runs**, tick **Reference** for four orders the shift considered good.
3. Tick **Compare** for yesterday's slow order.
4. In **Signals**, search `Palletiser/` and **Choose every match**, then add `Packaging/Checkweigher/Rate`.
5. Click **Compare**. Read the warnings first.
6. In **Deviations**, sort by **Outside band**. A palletiser signal with a large **Worst** at phase `0.8` and good **Coverage** says the order slowed near its end, at the palletiser.

Administration in the browser

> **Module: Web HMI.** This chapter covers the administration screens of the browser client and its first-run setup page. The top bar, the command palette and the status bar they sit in are described in *The browser client: finding your way*. It is present only on installations licensed for the Web HMI module. The Desktop offers the same administration in every installation; the base chapters *Users and security*, *Licensing and modules*, *The historian* and *Configuration and deployment* describe it there.

A site run entirely from browsers — a Web-only installation, a containerised one, or a plant server nobody logs in to — has to be administered without a Desktop and without a shell on the server. The **Administration** menu in the browser's top bar is that administration: people, integration accounts, stations, single sign-on, the audit trail, modules, the licence, branding, certificates, I/O, tag drift, the historian, run sources, linked nodes, the fleet and the Studio Assistant.

Everything on these screens belongs to the **node**. The browser asks the node with the administrator's own session; the node decides whether they may, makes the change, and records it in its own trail under that person's name. A configuration change made here (I/O, the historian, a module switched on) becomes a numbered **configuration revision**, exactly as one made on the Desktop does, and can be compared and rolled back.

The examples use the demonstration packaging line: node **Web Node**, connection `Packaging` on the simulator, thirteen acquisition groups (`Erector`, `BottomSealer`, `Loader`, `Checkweigher`, … `CasePacking`), and the seeded accounts `admin` (Administrator) and `shift` (Operator).

Reaching the administration screens

The **Administration** menu (top bar, with a gear icon) lists the screens in two groups. An administrator sees all sixteen. Anybody else sees only the screens the node lets them use: **I/O** and **Tag drift** for a person the node lets change its I/O, and **Historian** for one it lets configure the historian. With none of them, the menu is not drawn. The server refuses every screen to anybody it does not allow, whatever the menu shows: a person who types the address of a screen that is only for administrators reads "You are not permitted to administer this installation."

Group	Screen	Address
This installation	**Users & permissions**	`/admin/users`
	Service accounts	`/admin/service-accounts`
	Stations	`/admin/stations`
	Single sign-on	`/admin/single-sign-on`
	Audit trail	`/admin/audit`
	Modules	`/admin/modules`
	License	`/admin/license`
	Branding	`/admin/branding`
What it talks to	**Certificates**	`/admin/certificates`
	I/O	`/admin/io`

Group	Screen	Address
	Tag drift	`/admin/drift`
	Historian	`/admin/historian`
	Run sources	`/admin/run-sources`
	Linked nodes	`/admin/linked-nodes`
	Fleet	`/admin/fleet`
	Studio Assistant	`/admin/assistant`

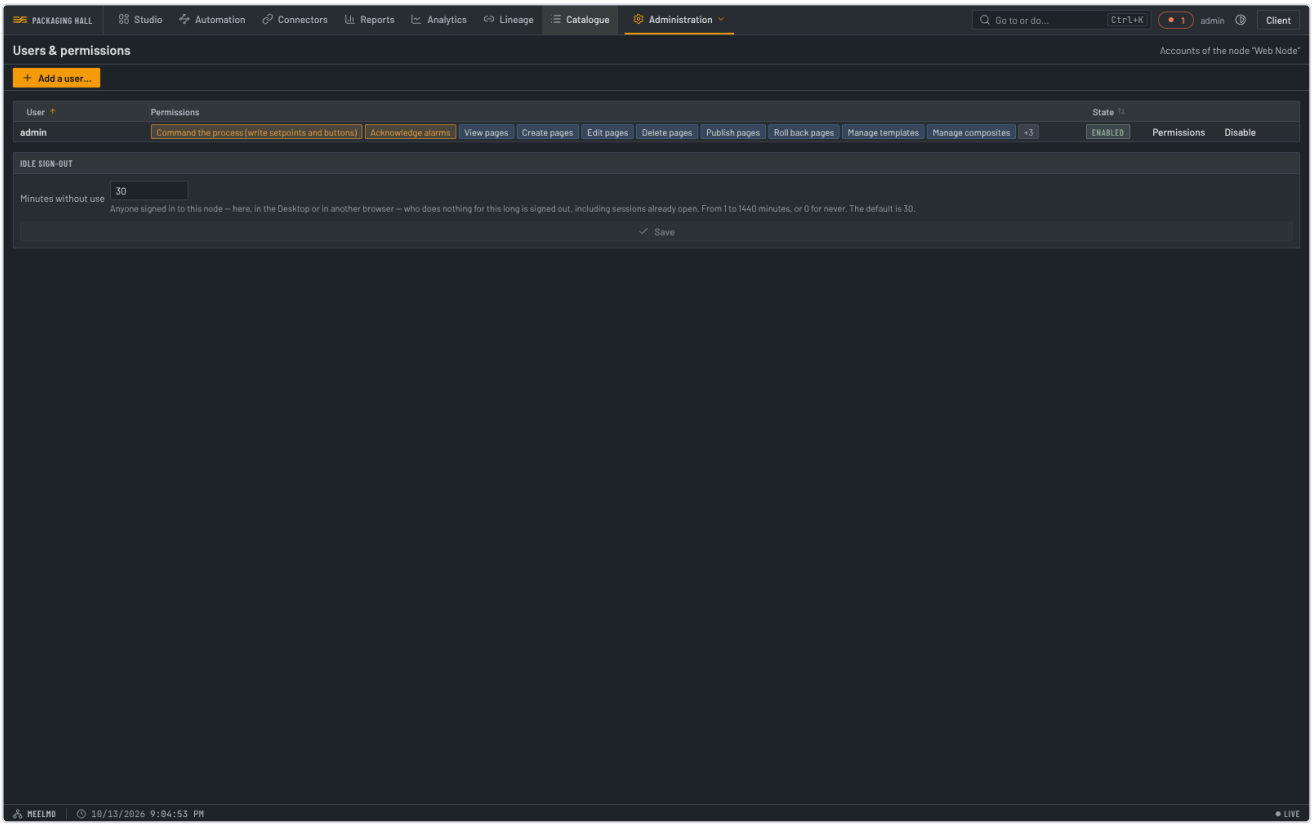
The first group is about who may use the installation; the second about what the installation talks to. A review page, **Interface primitives** (`/admin/interface`), is reachable by address only and is described at the end of this chapter.

The same screens are in the command palette (**Ctrl+K**), in the phone's **Go to** sheet and, on the operator screens, under **⋯**. **Alt+1** to **Alt+9** reach the first of them directly; each menu line shows its shortcut. *The browser client: finding your way* describes the palette, the jumps and the phone sheet.

Lineage is not an administration screen: it is in the top bar for everybody, and is described in *Reports and analysis in the browser ▸ Lineage*.

Users & permissions

Administration ▸ Users & permissions lists the people who may sign in and what each may do. The strip under the title says whose accounts these are — "Accounts of the node “Web Node”", "Accounts of this installation's node", or "Accounts of this installation" where there is no node.



Users & permissions with the admin account: its permissions as a line of tags, the two that command the process first and in orange, its state Enabled, and the Permissions and Disable buttons on the row.

Column	Shows
User	display name in bold, with the username beside it when different. Sorts
Permissions	one line of tags; those that do not fit are counted (`+3`) and all are listed on hover. **Command the process (write setpoints and buttons)** and **Acknowledge alarms** always come first and are orange
State	**Enabled** or **Disabled**
row buttons	**Permissions** (edit) and **Disable** / **Enable**

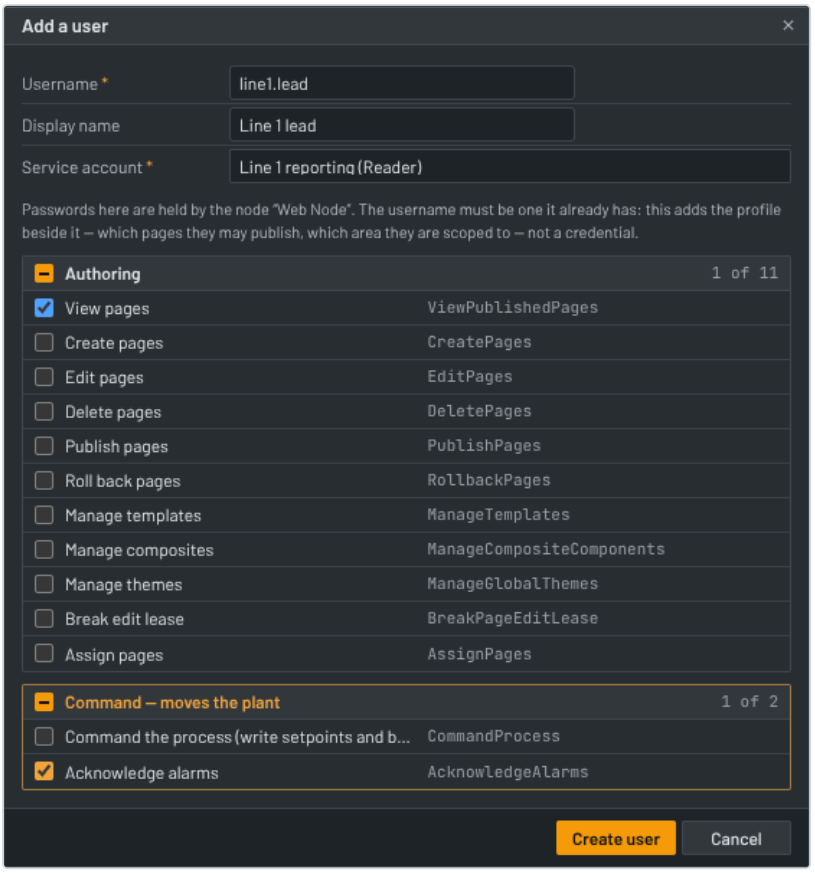
The row menu (the **⋯**, a right click, or the Menu key) holds **Edit permissions…**, **Disable** or **Enable**, **Copy username**, **Show in the audit trail** (opens the audit trail filtered to that username) and **Reset password…**, which is greyed out with an explanation where the node holds the passwords: "The node “Web Node” holds the passwords here: reset this one there."

Adding somebody

Add a user… opens a dialog:

Field	Meaning
Username (required)	The sign-in name
Display name	What the shell and the audit trail show
Service account (required)	The service account whose permissions, signal scope and limits govern what this person can read of the plant ("— choose an account —"). Several people may share one
Password (leave blank to generate one)	Only on an installation that holds its own passwords. Left blank, a password is generated and shown once: "Generated password, shown once:"
Authoring permissions	**View pages**, **Create pages**, **Edit pages**, **Delete pages**, **Publish pages**, **Roll back pages**, **Manage templates**, **Manage composites**, **Manage themes**, **Break edit lease**, **Assign pages**
Command — moves the plant	**Command the process (write setpoints and buttons)**, **Acknowledge alarms** — kept apart and coloured so that nobody grants authority over the plant by ticking the next box down

Where the node holds the passwords, the dialog says so: the username must be one the node already has, and this adds the profile beside it (which pages the person may publish, which area they are scoped to), not a credential.



Add a user filled in for line1.lead, display name Line 1 lead, service account Line 1 reporting: the note that the node "Web Node" holds the passwords here, the Authoring list with View pages ticked, and Command — moves the plant with Acknowledge alarms ticked.

Create user creates the account; errors (for example "A username and a service account are required.") appear in the dialog.

Changing, disabling and enabling

- **Permissions** on a row (or **Enter**) opens **Permissions — <name>** with the same two lists. **Save** stores the change: "Permissions saved for shift."
- **Disable** asks first — "Disable shift? They cannot sign in until the account is enabled again. Their permissions are kept." — because it locks a person out mid-shift.
- **Enable** gives the account back.

Example: let the Operator acknowledge but not command. Open the `shift` row's **Permissions**, tick **View pages** and **Acknowledge alarms**, leave **Command the process** unticked, and **Save**. The row shows **Acknowledge alarms** first, in orange, followed by **View pages**.

Deleting somebody

Delete… on the toolbar (with one or more rows selected — Ctrl-click or Shift-click) or in a row's menu asks once, naming the people: *Delete Shift lead?* On an installation whose node holds the accounts, the question says that only the profile here goes — its permissions and home page — and that the account they sign in with is the node's and stays (remove it in the Desktop's Users window). Disabling somebody is the gentler choice and is undone with one click.

The server refuses two deletions and the page says why: **your own account** ("You cannot delete the account you are signed in with") and **the last person who may administer** the installation. Each deletion and each refusal is in the audit trail as `auth.user.deleted`.

Idle sign-out

Where the accounts are a node's, a panel under the table, **Idle sign-out**, sets how long somebody signed in to the node may do nothing before they are signed out. It is the node's setting, so it applies to everybody signed in to the node: in this browser, in any other browser on the node, and at the Desktop.

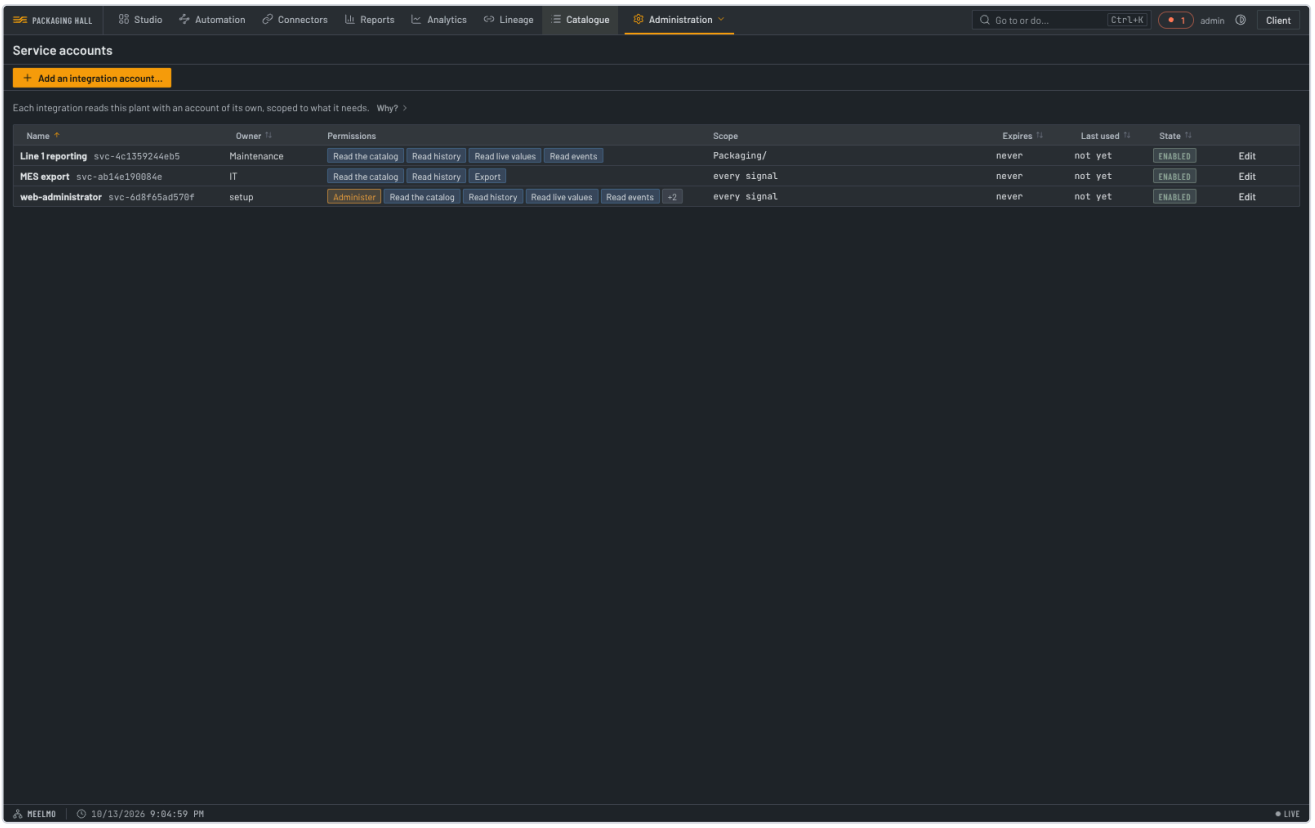
Field	Meaning
Minutes without use	From 1 to 1440; a larger number is held at 1440 (a day). **0** means never. The default is 30
Save	Stores the value at the node: "The node's idle sign-out was saved." Greyed out until the number differs from what the node holds

While the box holds 0 the panel warns: "0 means never. That departs from the security baseline, which signs idle sessions out after 30 minutes."

The panel is shown to somebody the node granted **Users.Manage** (the Administrator role), and only they may change it — the node checks again, whatever the page shows. The value is read and saved at the node as the person signed in here, on a host that holds the node and on a plant server, where this host is attached to a node the recorder service holds; each change is in the node's audit trail as `security.session_idle`, with the old and the new value and who made it. The change applies at once, including to people who are already signed in: a browser is signed out at its next request once it has been idle for longer than the new value. With **0**, a browser session still ends after 12 hours without use, which is the browser's own limit. On an installation without a node, or with a node too old to hold the setting, the panel is not shown.

Service accounts

Administration ▸ Service accounts manages the accounts integrations read the plant with — a reporting tool, an export, a dashboard. "Each integration reads this plant with an account of its own, scoped to what it needs." They belong to the node, which records who created or changed each one.



Service accounts with the web-administrator account created by setup: its owner, its permissions led by Administer, its scope every signal, never expiring, not yet used, enabled.

Column	Shows
Name	display name, with the account id (`svc-…`) beside it; the description on hover
Owner	who to ask about it
Permissions	tags: **Administer**, **Read the catalog**, **Read history**, **Read live values**, **Read events**, **Export**, **Manage datasets**, **Write annotations**. A permission the account holds that this screen does not offer is shown too, and kept on every save
Scope	the signal prefixes it may read (**every signal** when unrestricted), then **Never:** and the denied prefixes
Expires	the last day its token works, or **never**
Last used	when it last authenticated, or **not yet**
State	**Enabled** or **Disabled**
Edit	opens the account

An account with **Manage datasets** saves dataset definitions — and their drops, question synonyms, process limits and cross-checks — wherever they are kept, including where the node keeps them: the node checks the account's own permission, lets a dataset name only signals the account may read, and records the revision under the account (`svc-…`) with its name and owner. A web host attached to the node passes the token on, so the save is the account's there too and never the host's.

On a plant server, the account the web host reads its node with is listed here too, with a **WebHost** tag. It grants nothing to read or change; it tells the node the requests are every browser's on that host rather than one integration's, so the node allows it a host's request budget instead of one client's. Setup gives it

to the account it makes for a web host joining a node; the host keeps each person it serves to their own budget, and when a budget is spent a page says so and when to try again (`429` with `Retry-After`) rather than reporting an error in the host.

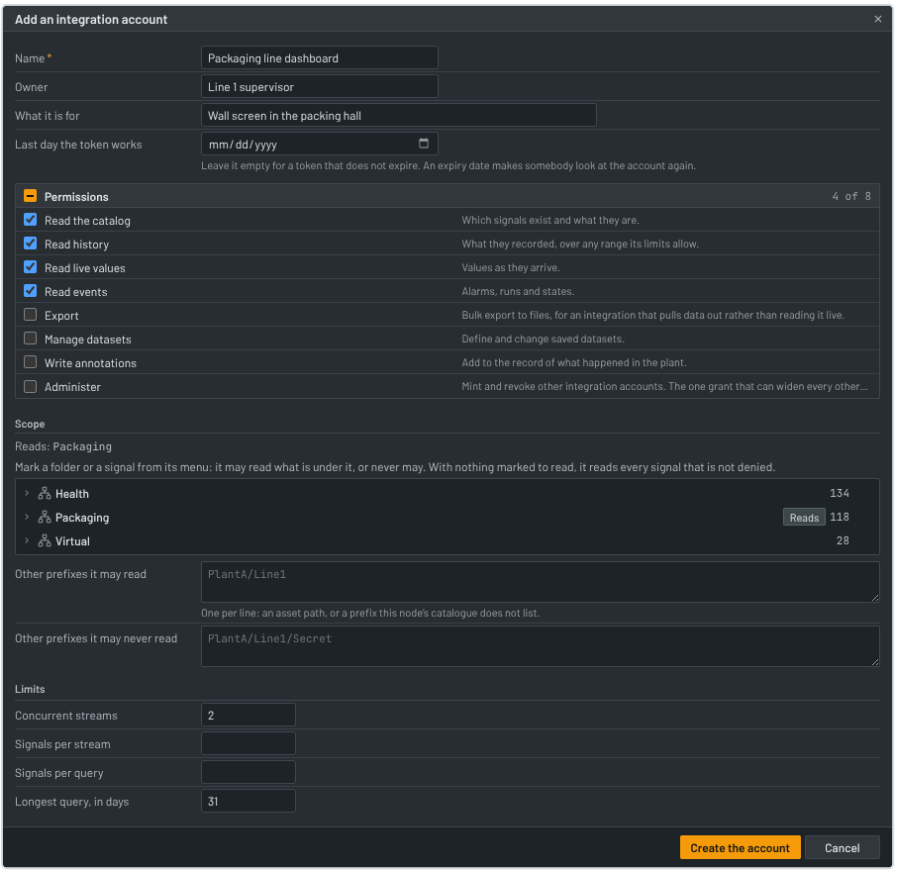
The row menu holds **Edit…**, **New token…**, **Disable** / **Enable** and **Delete…**. The three that break an integration ask first:

- **New token…** — "Issue a new token for …? The token the integration has now stops working at once; it needs the new one."
- **Disable** — "Its token is refused until the account is enabled again. Nothing about the account is lost."
- **Delete…** — "Its token stops working at once, and this cannot be undone."

Adding an integration account

Add an integration account… opens the dialog. A new account starts with the four read permissions ticked, because most integrations are readers.

Field	Meaning
Name (required)	a name somebody will recognise, for example `Packaging line dashboard`
Owner	who to ask about it ("Who to ask about it")
What it is for	a description
Last day the token works	leave empty for a token that does not expire; an expiry date makes somebody look at the account again
Permissions	a checklist, each with a sentence: **Read the catalog** (which signals exist and what they are), **Read history**, **Read live values**, **Read events** (alarms, runs and states), **Export** (bulk export to files), **Manage datasets**, **Write annotations**, **Administer** (mint and revoke other integration accounts — the one grant that can widen every other)
Scope	the node's signal catalogue as a tree. Right-click a folder or a signal and choose **It may read this**, **It may never read this** or **Clear the mark**; marked rows show **Reads** or **Never**. With nothing marked to read, it reads every signal that is not denied
Other prefixes it may read, **Other prefixes it may never read**	one prefix per line, for an asset path or a prefix the catalogue does not list. Where the catalogue cannot be read, these two boxes are the whole scope editor
Limits	**Concurrent streams** (1–64), **Signals per stream**, **Signals per query**, **Longest query, in days** (1–3660); blank takes the server's default



Add an integration account filled in as the example below: Packaging line dashboard, owner Line 1 supervisor, Wall screen in the packing hall; the four read permissions ticked; Scope reading Packaging, marked Reads in the tree beside Health and Virtual; Concurrent streams 2 and Longest query, in days 31.

Create the account creates it and opens **The token for …** — once. "Copy it now and give it to the integration. It is shown this once and can never be shown again; if it is lost, rotate the account for a new one." **Copy the token** copies it; **I have it — hide it** closes the dialog and the page forgets the token. The node keeps only a digest.

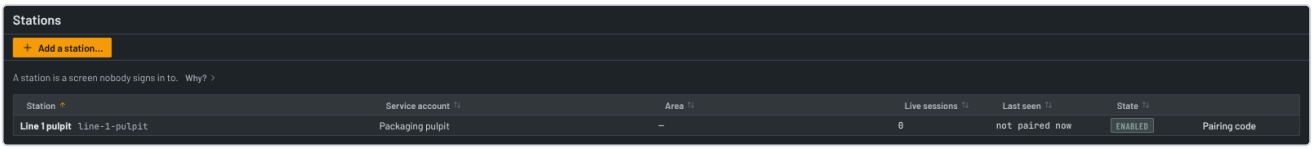
Example: a read-only account for the line dashboard.

1. **Add an integration account…**
2. **Name** `Packaging line dashboard`, **Owner** `Line 1 supervisor`, **What it is for** `Wall screen in the packing hall`.
3. Leave the four read permissions ticked.
4. In **Scope**, right-click `Packaging` ▸ **It may read this**; open it and right-click `Packaging/Checkweigher/WeightTarget` ▸ **It may never read this**.
5. **Limits**: **Concurrent streams** `2`, **Longest query, in days** `31`.
6. **Create the account**, **Copy the token**, paste it into the dashboard's configuration, then **I have it — hide it**.

The data-access chapter of the Data access and SDKs module, where licensed, describes how an integration uses the token.

Stations

Administration ▸ Stations manages the fixed terminals that show the plant without anybody signing in — the panel beside a machine, the screen on a control-room wall. A station acts as a service account once it is paired, so make the account first, under **Service accounts**.



Stations with Line 1 pulpit: its id line-1-pulpit, the service account Packaging pulpit, no area, no live sessions, not paired now, Enabled, and Pairing code on its row.

Column	Shows
Station	the name in bold, with the station id beside it. Sorts
Service account	the account the station reads and acts as at the node
Area	the area its assignments are made for, or **—**
Live sessions	how many screens are paired as it now. **—** when the host would not list sessions: the count is then unknown, not zero
Last seen	when one of its screens was last heard from, or *not paired now*
State	**Enabled** or **Disabled**. A disabled station's row is greyed
row button	**Pairing code**, greyed with the reason when a code cannot be issued

The row menu (the **⋯**, a right click, or the Menu key) holds **Edit** (also **Enter** on the row), **Issue a pairing code**, **End its sessions…** and **Delete…**.

Adding a station

Add a station opens a dialog; **Edit** opens the same one as **Editing <name>**.

Field	Meaning
Station id (required)	Letters, digits, dots, hyphens and underscores, starting with a letter or a digit, at most 64 characters, such as `line-1-pulpit`. It is what a code, a session and the audit trail name the station by, so it cannot be changed later
Name (required)	What the list shows
Service account (required)	"The station reads and acts at the node as this account." Choose from the integration accounts (— *choose an account* —)
Roles	Page assignments made for these roles apply to this station, as they do for a person. Separate them with commas. A new station starts with `Operator`
Area	Page assignments made for this area apply to this station, for example `Plant/Line1`. Blank for none
Language	The language its screens open in when it is paired, each language in its own name. Nobody signs in at a station to choose one, so this is the only place it is set. English for a new station
State	**Pairing codes can be issued for it**. Ticked for a new station. Cleared, the station is **Disabled** and no code can be issued for it until it is ticked again

On its screens sets what the station's screens may do. It is used while the service account holds no screen grant at the node; once the account holds one, the node's grant applies instead.

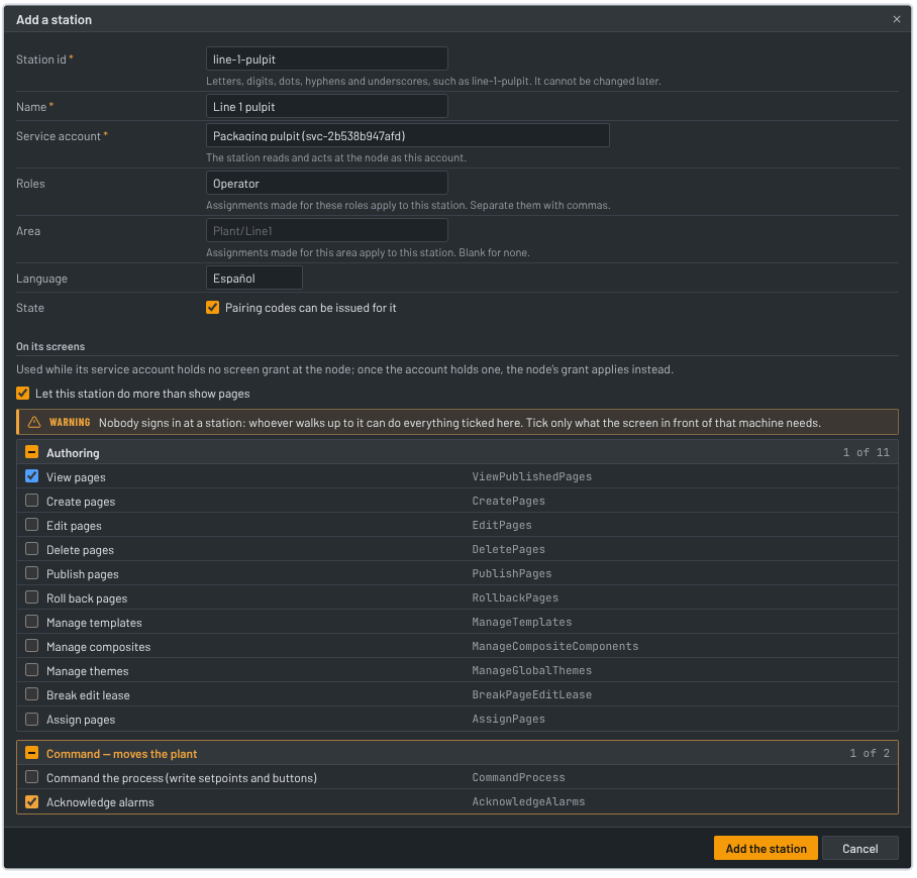
- By default the section says *It shows published pages and does nothing else.*

- Tick **Let this station do more than show pages** to offer the same two lists as **Users & permissions**: the authoring permissions (**View pages** … **Assign pages**) and, in orange, **Command the process (write setpoints and buttons)** and **Acknowledge alarms**. The warning beside it says why to be careful: "Nobody signs in at a station: whoever walks up to it can do everything ticked here. Tick only what the screen in front of that machine needs."

Add the station (or **Save** when editing) stores it: "Line 1 pulpit was added."

Example: a pulpit screen for the packaging line.

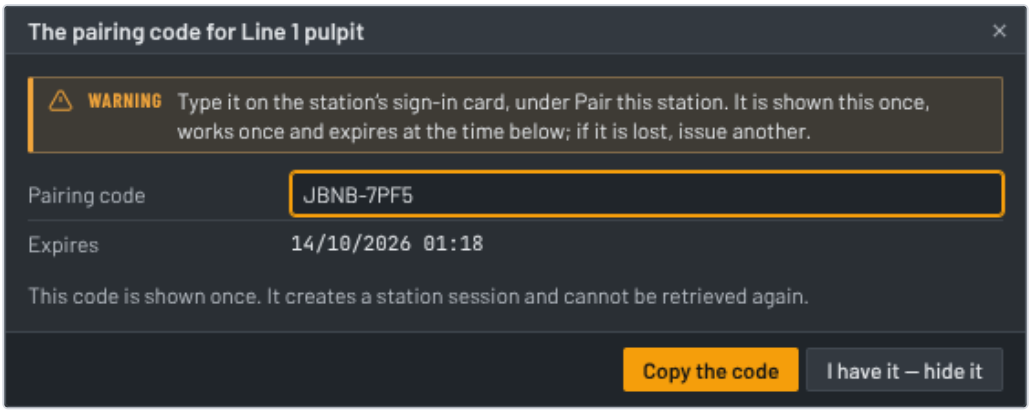
1. Under **Service accounts**, add an account `Packaging pulpit` with **Read live values** and **Read events**, scoped to `Packaging`.
2. Under **Stations**, **Add a station**: **Station id** `line-1-pulpit`, **Name** `Line 1 pulpit`, **Service account** `Packaging pulpit`, **Roles** `Operator`, **Area** blank, **Language** `Español` for the Spanish-speaking shift.
3. Tick **Let this station do more than show pages**, then tick **View pages** and **Acknowledge alarms** only.
4. **Add the station**, then **Pairing code** on its row.



Add a station filled in as the example: station id line-1-pulpit, name Line 1 pulpit, service account Packaging pulpit, role Operator, no area, Language Español and Pairing codes can be issued for it; under On its screens, Let this station do more than show pages ticked, the warning, View pages and Acknowledge alarms ticked.

Pairing a screen

On the station's row, **Pairing code** issues a one-time code. It is shown once, with the time it expires — a few minutes later — and **Copy the code** puts it on the clipboard. On the station's browser, open the sign-in card's **Pair this station** tab and type the code. **I have it — hide it** closes the code; if it is lost, issue another. A disabled station, or one whose service account is missing or disabled, is not offered a code, and the row says why.



The pairing code for Line 1 pulpit: the warning to type it under Pair this station, the code, when it expires, Copy the code and I have it — hide it.

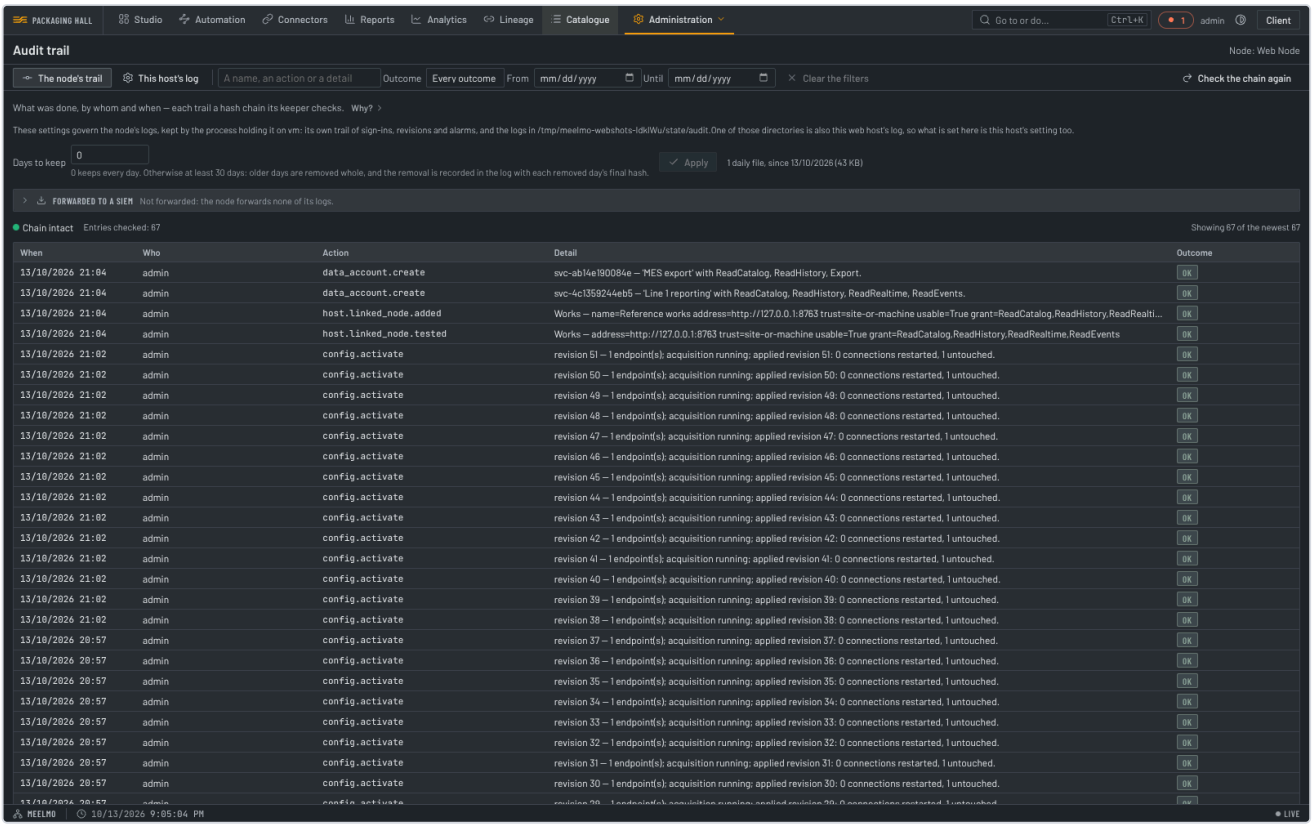
Ending its sessions and deleting it

From the row's menu, **End its sessions…** signs out every screen paired as the station at once; they show the pairing card again and the profile stays. **Delete…** does the same and removes the profile. Both ask first, and every one of these actions — saving, issuing a code, ending sessions, deleting — is recorded in the audit trail.

Audit trail

Administration ▸ Audit trail answers "what was done, by whom and when". There are two trails, and each is a hash chain its keeper checks:

- **The node's trail** records what reached the node from any screen — configuration revisions, alarm acknowledgements and shelving, accounts.
- **This host's log** records what happened on this web host — sign-ins, signatures, writes sent from this host.



The node's trail: forty-nine entries checked and the chain intact, most of them config.activate by admin, each with its detail and the outcome OK.

Control	Does
The node's trail / **This host's log**	chooses the trail
Contains ("A name, an action or a detail")	keeps entries whose user, action or detail contains the text
Outcome	**Every outcome**, **Done**, **Refused**, **Failed**
From, **Until**	a date range
Clear the filters	removes all filters
Check the chain again	walks the chain again

The line under the toolbar gives the chain's verdict — **Chain intact**, **Chain broken — an entry was edited, removed or inserted** (with "Where it breaks: …"), or **Not checked** — and how many entries (node) or daily files (host) were checked. On the right: "Showing 49 of the newest 49". Entries written before the node kept a chain are counted separately.

Column	Shows
When	date and time
Who	the user
Role	host log only
Action	for example `config.activate`, `auth.sign_in`, `data_account.create`
Detail	full text on hover, for example "revision 37 — 1 endpoint(s); acquisition running."
Outcome	**OK**, refused or failed

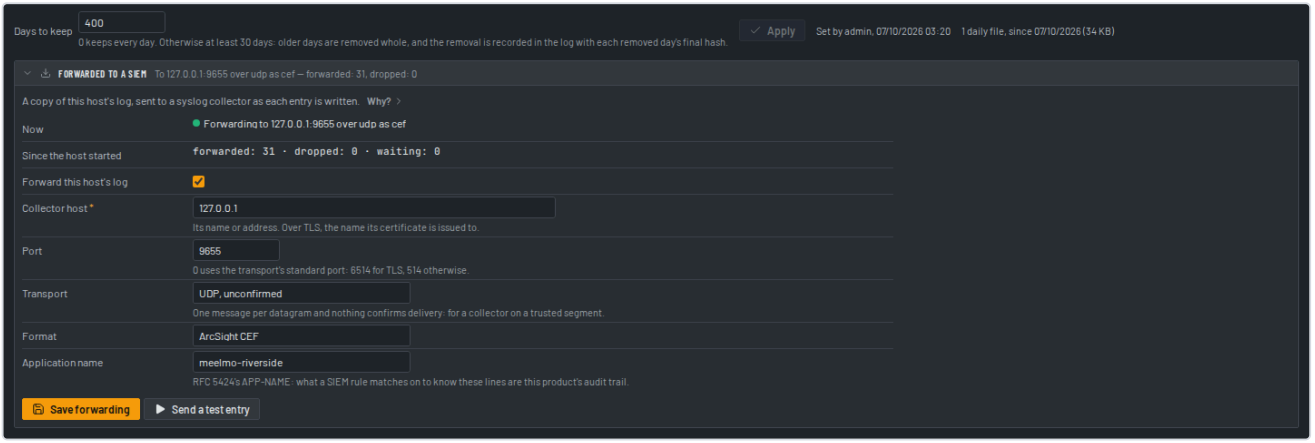
A row's menu offers **Only entries by <user>** and **Only entries with <action>**.

With **This host's log** chosen, **How long this host keeps its log** appears: **Days to keep** (0 keeps every day; otherwise at least the shortest period stated) and **Apply**. Older days are removed whole and the removal is recorded with each day's final hash. The line beside it says who set it and when, and how many daily files are held. When the environment variable `MEELMO_AUDIT_RETENTION_DAYS` is set on the machine it wins, and the field is locked.

A line above the settings says which log they govern, on which machine and in which directory.

With **The node's trail** chosen, the same two settings govern **the node's logs**, and an administrator at the node (the permission to manage users) sets them there: on a plant server those are the recorder service's, which this host's own settings never reached. The node's process writes three trails — its own, in its database (sign-ins, revisions, alarms, everything done at the node as a person), the log in its state directory, and its data API's log (every read any account makes, which is the one that grows; `api-state/audit` beside the recorder as installed). **How long the node keeps its logs** applies to every log directory the node's process writes; its own trail, in its database, is never pruned. **Forward the node's logs and its trail** writes `forwarding.json` into each of those directories and is in force at the node at once, and the node forwards its own trail with them. The line above the settings names the machine and the directories, and says when one of them is also this host's log — on a plant server as installed, this host's log is the recorder's state directory, so a setting made there is this host's too. Anybody else sees the trail without the settings; an engineer is refused by the node.

Under it, **Forwarded to a SIEM** sends a copy of this host's log to a syslog collector as each entry is written: **Forward this host's log**, **Collector host**, **Port** (0 is the transport's standard port), **Transport** (**TLS (RFC 5425)**, **TCP, in the clear** or **UDP, unconfirmed**), **Format** (**RFC 5424 with structured data** or **ArcSight CEF**) and **Application name**. It is best-effort and after the file: a collector that is slow or gone never holds the trail on disk back. The settings are kept in `<state>/audit/forwarding.json`; saving them here is in force on this host at once, and **Now** says what the running forwarder does — where it sends, and how many entries it has forwarded, dropped and still holds — beside what the file says. Another process writing the same directory reads the file when it starts; set the node's logs under **The node's trail** to apply it there at once. **Send a test entry** records a real entry, labelled as a test, so it can be found at the collector.

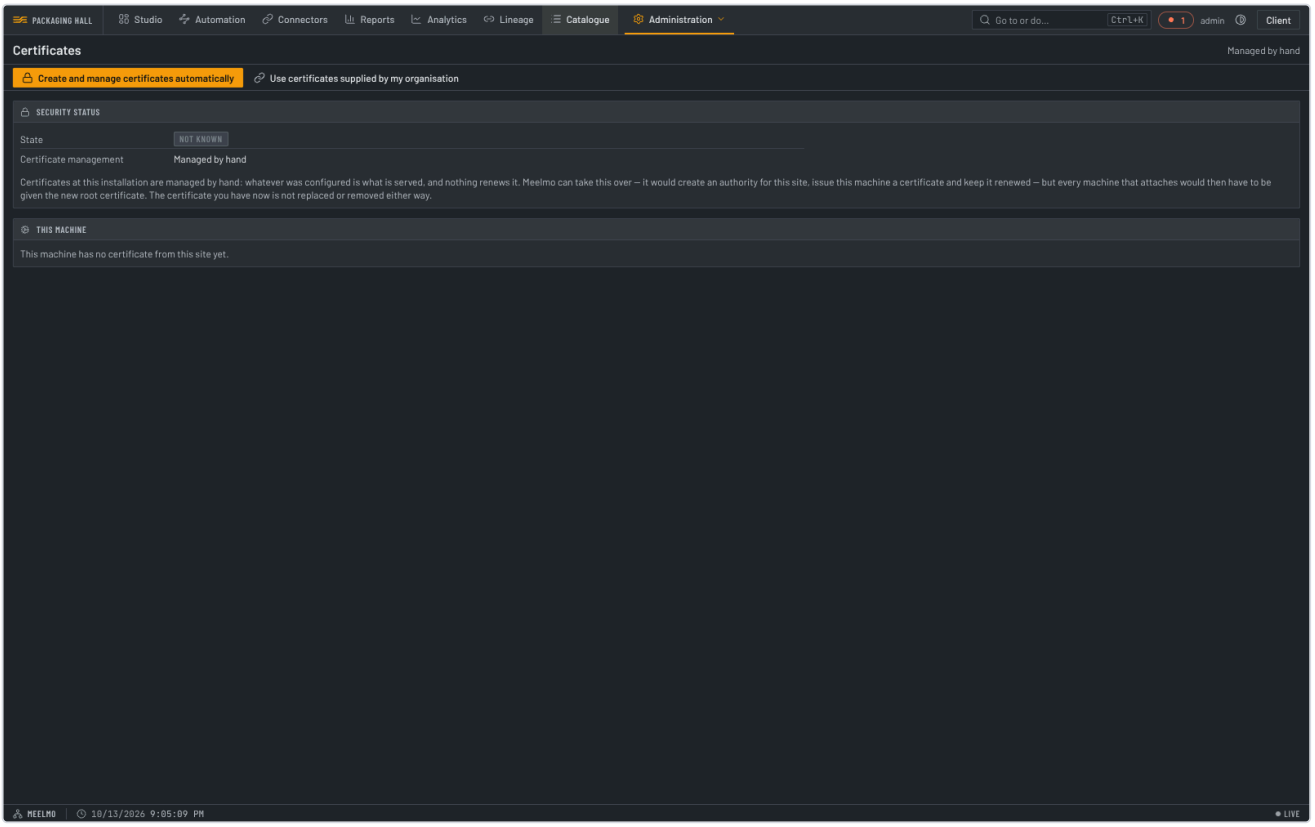


This host's log on the reference works: kept for 400 days, set by admin, and Forwarded to a SIEM — forwarding to 127.0.0.1:9655 over UDP as ArcSight CEF, with the application name meelmo-riverside and the counts since the host started

Example. To see who changed the I/O yesterday: choose **The node's trail**, type `config.activate` in **Contains**, set **From** and **Until** to yesterday, and read **Who** and **Detail**.

Certificates

Administration ▸ Certificates is the site's certificate authority, from a browser. The strip under the title states how this site manages certificates: **Managed by hand**, **Meelmo-managed site PKI** or **Your organisation's PKI**.



Certificates on an installation managed by hand: the two buttons to adopt a managed PKI or use the organisation's certificates, the security status Not known, and This machine without a certificate from this site.

The toolbar

Button	Shown when	Does
Create and manage certificates automatically	not already Meelmo-managed	Creates an authority for this site, issues this machine a certificate and keeps it renewed. Every machine that attaches must then be given the new root certificate. The certificate in use now is not replaced or removed
Use certificates supplied by my organisation	not already organisation-managed	Hands issuance back to your organisation's authority; Meelmo will not renew what it cannot sign
Export root certificate…	a root exists	Shows the root certificate (PEM) to copy. It is public by construction; private keys are never exported
Replace the issuing CA	a root exists	Replaces the issuing authority under the same root, so nothing needs redistributing
Renew now	Meelmo-managed	Renews this machine's certificate now
Invite a machine…	Meelmo-managed	Creates an enrolment invitation (below)
Create a signing request…	organisation-managed	Produces a signing request (PEM) for your authority; the private key stays on this machine

The panels

- **Security status** — **State** (**Healthy**, **Renewal due**, **Renewing**, **Expiring soon**, **Expiring — act now**, **Renewal failed**, **Expired**, **Withdrawn** or **Not known**), **Certificate management**, **Site**, and for a Meelmo-managed site **Renewal** ("Certificates last 90 days and renew themselves with 30 days left. Nothing needs doing."). A failed renewal is shown in red above the panels; the certificate already in use is still served.
- **Site authorities** — **Root CA** and **Issuing CA**, each with **Subject**, **Expires**, **Thumbprint** and **State**.
- **This machine** — **Subject**, **Expires** (with days left), **Thumbprint** and **Names and addresses**. A certificate is accepted only for the names in it, so "it will not connect" is usually this list not containing what somebody typed. Without one: "This machine has no certificate from this site yet."
- **Enrolled machines** (Meelmo-managed) — **Machine**, **Names and addresses**, **Expires**, **State** (**Valid** or **Withdrawn**), with the number of outstanding invitations. The row menu's **Withdraw…** asks first; a withdrawn machine can no longer attach, and re-enrolling needs a new invitation and a new key.

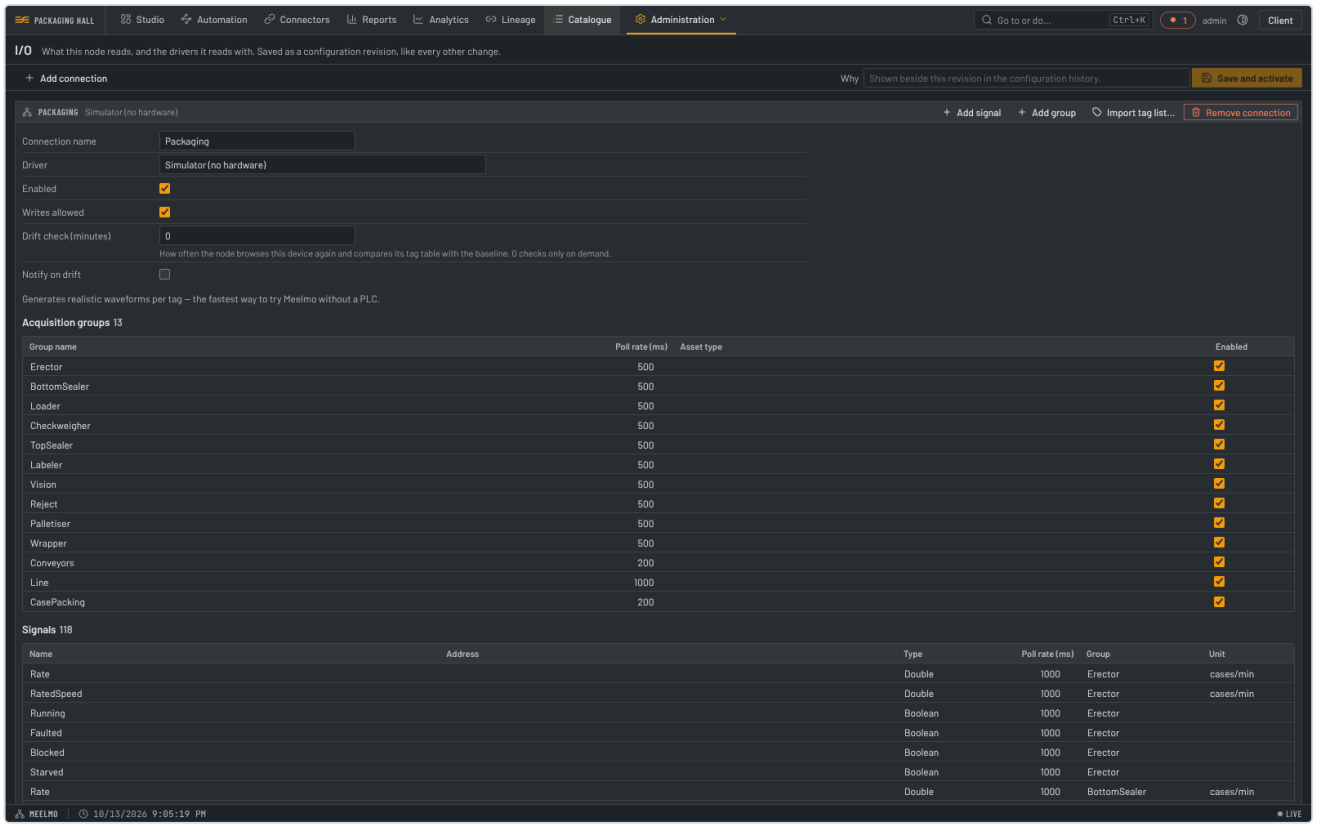
Invite a machine asks for **Machine name**, **Node id** and **Names and addresses it is reached by**, then **Create invitation** shows an **Enrollment token** once. The site keeps only a hash; the token is worth one certificate for one machine and expires in two hours.

Example: enrol a second recorder. On a Meelmo-managed site, **Invite a machine…**, **Machine name** `Palletiser recorder`, **Names and addresses** `palrec01, 10.20.1.31`, **Create invitation**. Give the token to whoever sets up that machine within two hours. **Export root certificate…** and install the root on every browser that will open the HMI.

I/O

Administration ▸ I/O is "what this node reads, and the drivers it reads with". It is the first thing a Web-only site configures. Every save is a configuration revision.

Who may use it is who the node lets: reading it needs the node's permission to read configuration, saving and importing its permission to change I/O — an engineer as well as an administrator. The menu offers it to them and to nobody else; the account a browser reads with, which every browser of the installation shares, does not decide.



I/O for the Packaging connection on the simulator: connection name, driver, Enabled and Writes allowed, the thirteen acquisition groups with their poll rates, and the first of 118 signals with type, poll rate, group and unit.

The toolbar

Control	Does
Add connection	Adds a connection (named `Connection 1`, ...)
Why	A sentence shown beside this revision in the configuration history
Save and activate	Sends the whole configuration to the node, which validates it and activates it as a new revision ("Saved as a new configuration revision."). If the node refuses, the page lists why: "The node refused this configuration:"

An **Unsaved changes** marker appears while there are edits.

A connection

Each connection is a panel headed by its name and driver (for example **PACKAGING** *Simulator (no hardware)*), with **Add signal**, **Add group**, **Import tag list...** and **Remove connection** on the right.

Field	Meaning
Connection name	for example `Packaging`
Driver	the protocol driver; a driver this node cannot run is listed as "not on this node" so a site can tell "not supported" from "not installed here"
Acquisition	where the driver offers a choice: **Polling**, **Subscription** or **Cyclic stream**
Enabled	whether the connection is read

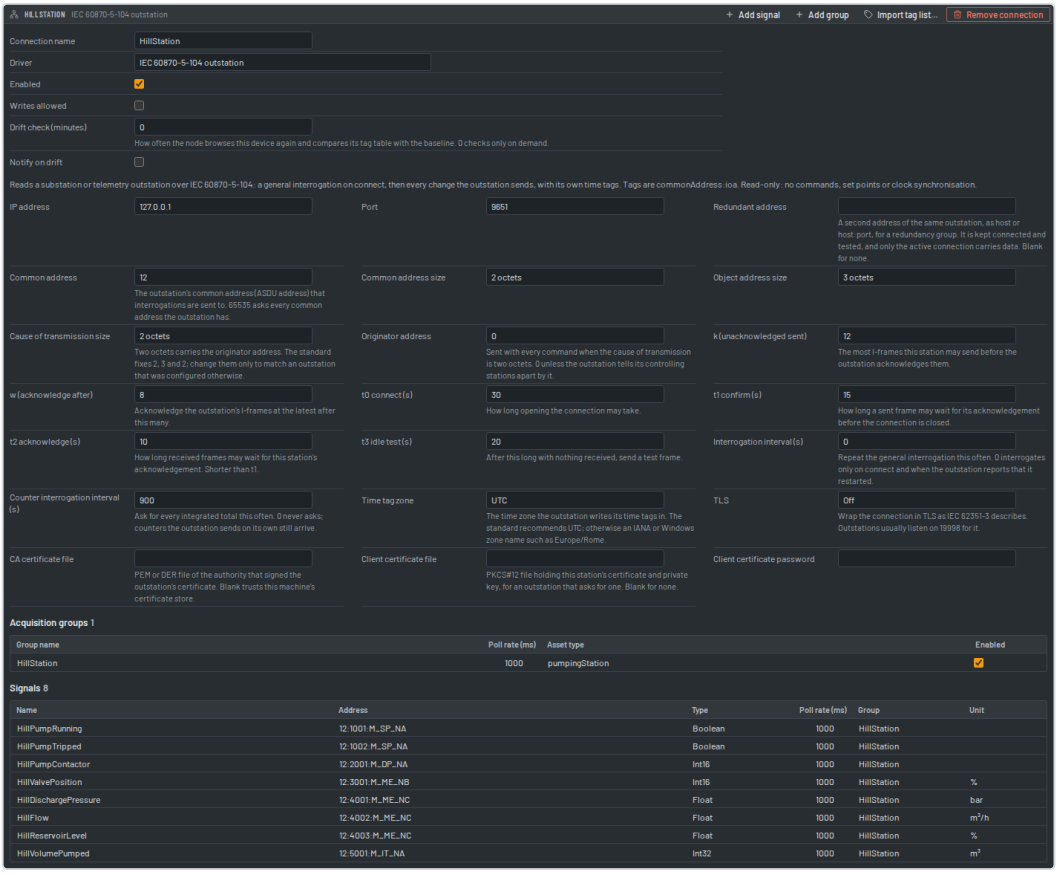
Field	Meaning
Writes allowed	whether commands from the HMI may write to this connection
Drift check (minutes)	how often the node browses the device again and compares its tag table with the baseline; `0` checks only on demand. See *Tag drift* below
Notify on drift	send a newly found drift once through the node's alarm notifications
driver settings	the fields the driver describes — address, port, rack and slot, security and so on. A secret already stored shows "unchanged"; type to replace it. "Leave blank for ..." states each default

Under it, a line describes the driver (the simulator "Generates realistic waveforms per tag — the fastest way to try Meelmo without a PLC.").

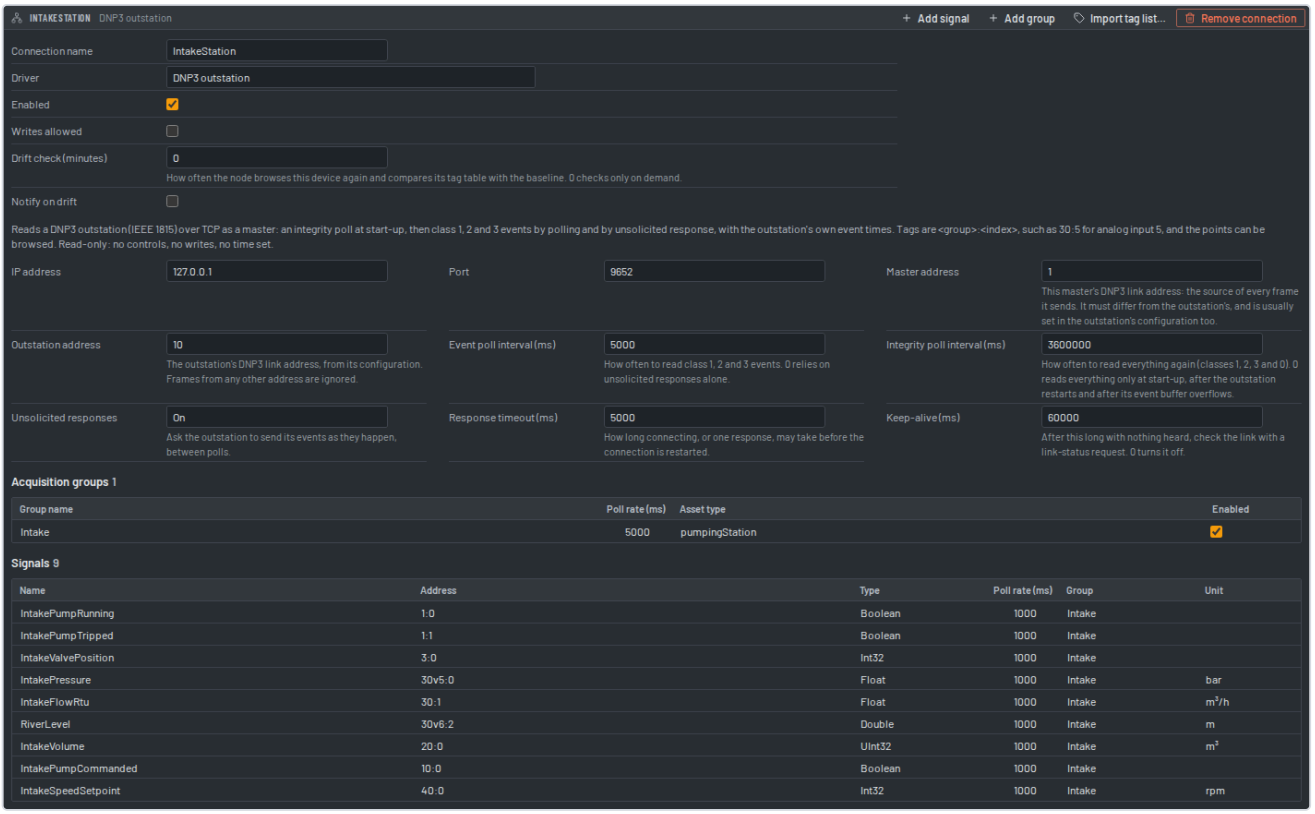
Acquisition groups — **Group name**, **Poll rate (ms)**, **Asset type**, **Enabled**. In the demonstration most groups poll at `500` ms, `Conveyors` and `CasePacking` at `200` ms and `Line` at `1000` ms.

Signals — **Name**, **Address**, **Type** (`Double` , `Boolean` , `String` , `Int32` ...), **Poll rate (ms)**, **Group** and **Unit**. For example `Rate` , `Double` , `1000` , group `Erector` , unit `cases/min` .

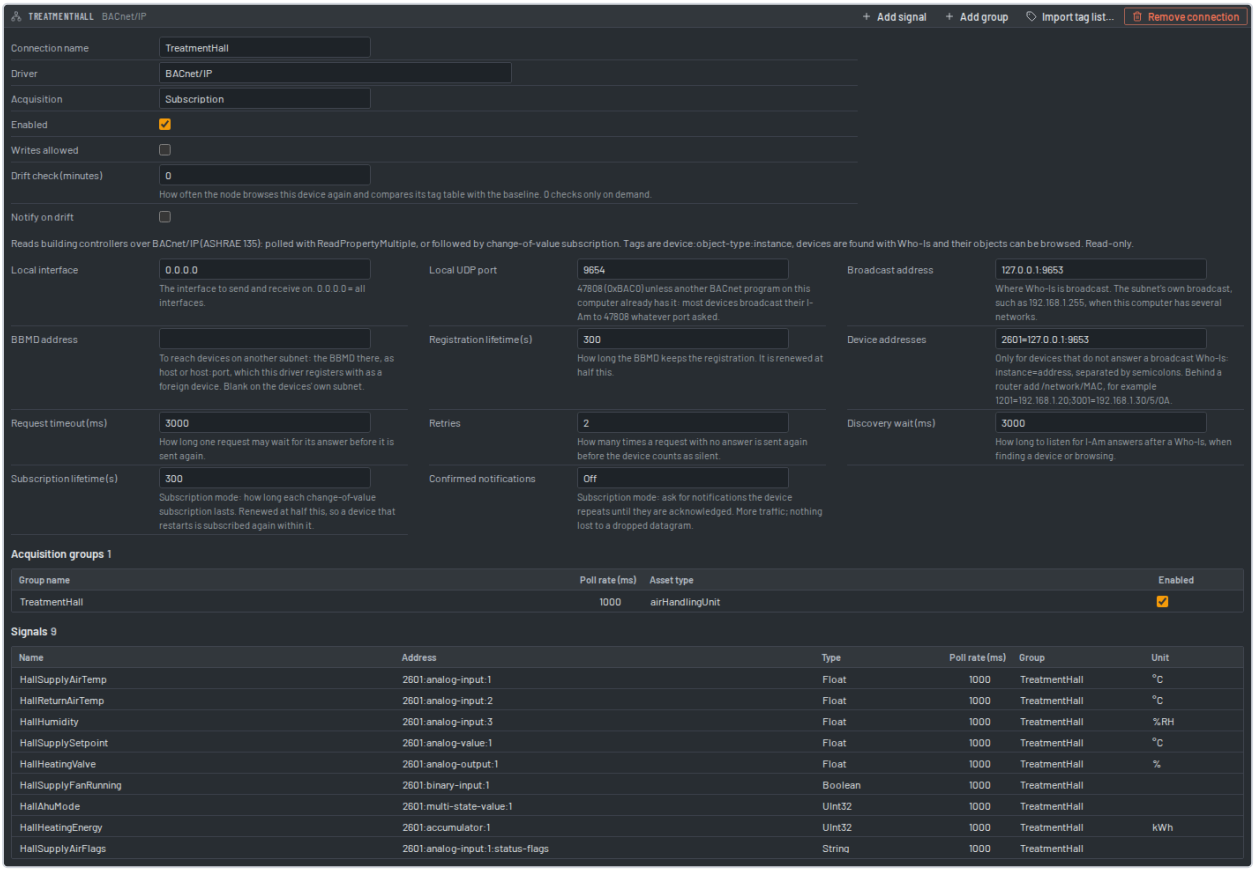
The utility protocols are connections like any other, each with the fields its driver describes. The reference works reads its hill pumping station over IEC 60870-5-104, its river intake over DNP3 and its treatment hall's air handling over BACnet/IP:



The HillStation connection: the IEC 60870-5-104 outstation driver, the outstation's address and port, common address 12, the field sizes, k and w, the t0 to t3 timers, the counter interrogation interval, the time-tag zone and TLS off — and its eight signals addressed commonAddress:ioa:type, from HillPumpRunning (12:1001:M_SP_NA) to HillVolumePumped (12:5001:M_IT_NA)



The IntakeStation connection: the DNP3 outstation driver with master address 1, outstation address 10, the event and integrity poll intervals and unsolicited responses on, and its nine signals addressed group:index — binary inputs, a double-bit input, analog inputs, a counter, and the status of a binary and an analog output



The TreatmentHall connection: the BACnet/IP driver following device 2601 by change-of-value subscription, its broadcast and device addresses, and nine signals addressed device:object-type:instance, among them analog inputs for the supply and return air, a multi-state value for the air handler's mode and an accumulator for the heating energy

Example: add a signal to the checkweigher.

1. On the `Packaging` connection, click **Add signal**. A row `Signal 1` appears.
2. **Name** `RejectRate`, **Address** as the driver expects it, **Type** `Double`, **Poll rate (ms)** `1000`, **Group** `Checkweigher`, **Unit** `cases/min`.
3. In **Why** type `Checkweigher reject rate for the shift report`.
4. **Save and activate**. The node activates a new revision; the audit trail shows `config.activate` with that revision number.

Before a save that breaks something

Save and activate first asks the host what the edit would reach. When it deletes, switches off, renames, retypes or moves a connection or a signal that something depends on, nothing is saved yet: a warning lists each change and what reads it — an alarm, a computed signal, a flow, a screen, a dataset, a report, a publication — marking the dependants that **would read nothing** afterwards. **Save and activate anyway** saves; **Keep editing** goes back to the screen. An edit nothing depends on saves straight away, as before.

On a host whose node is elsewhere the warning covers the project itself — acquisition, computed signals, alarms, detection rules and models — and says that the flows, screens, datasets, reports and publication could not be checked from here.

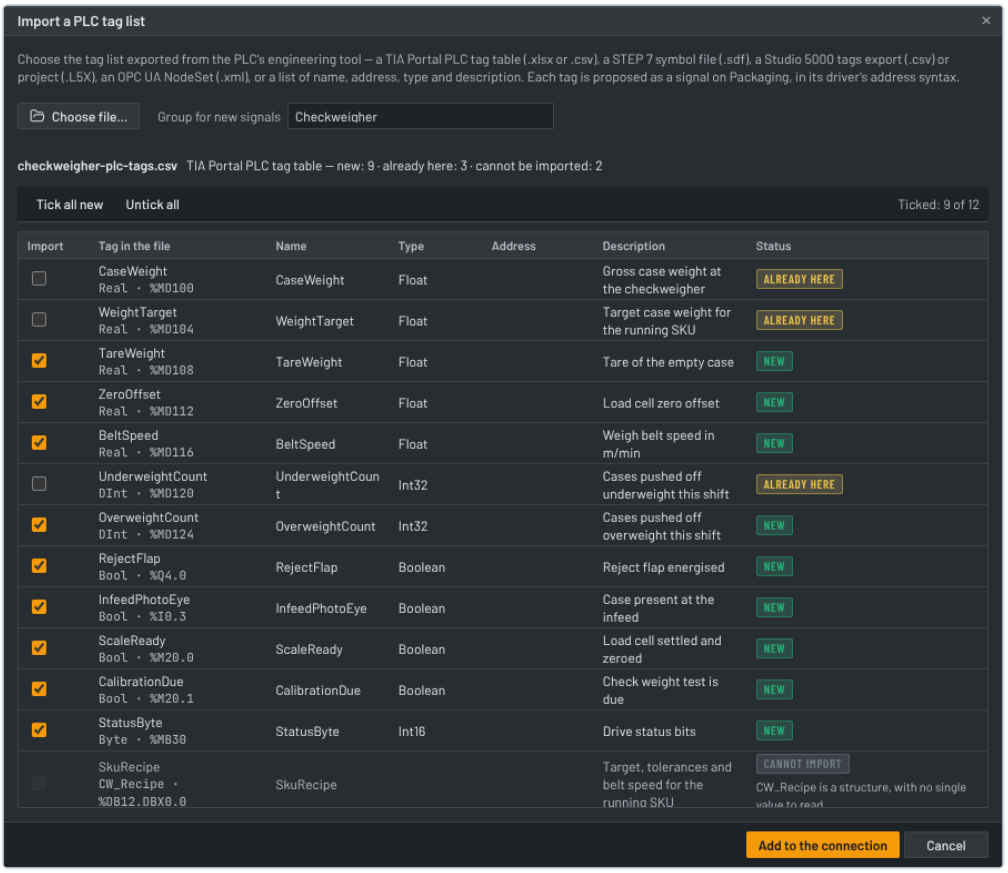
Importing a PLC tag list

Import tag list… on a connection reads the tag list the control engineer exported from the PLC's engineering tool and proposes each tag as a signal on that connection. It needs the same Administer permission as the rest of this screen. Which files it reads, and how each PLC type and address is translated for each driver, is in *Acquisition and I/O configuration ▸ Importing a PLC tag list*.

1. On the connection, click **Import tag list…** and **Choose file…**. For an OPC UA connection, first set **Namespace index** (3 for a Siemens S7-1500's own server). **Group for new signals** starts at the connection's first group.

2. The dialog names the file and what it is — for example *plc-tags.xlsx TIA Portal PLC tag table — new: 9 · already here: 1 · cannot be imported: 4* — and lists every tag: the tag in the file with its type and address, the proposed signal **Name**, **Type** and **Address**, the comment as **Description**, and a **Status**:

Status	Meaning	Ticked at first
New	No signal on the connection has this name	Yes
Already here	A signal has this name. Ticked, its address, type and description are replaced with the file's; its identity, and so its history, stays	No
Cannot import	The reason is shown beside the mark — *Byte is 8 bits, which this driver would read as 16*, *Recipe_UDT is a structure, with no single value to read*	Cannot be ticked



Import a PLC tag list on the Packaging connection, with the checkweigher PLC's TIA Portal tag table chosen and Checkweigher as the group for new signals: checkweigher-plc-tags.csv, a TIA Portal PLC tag table with nine new tags, three already here and two that cannot be imported; each row with its tag, type and address in the file, the proposed name and type, the comment as description and its status — CaseWeight, WeightTarget and UnderweightCount already here and unticked, TareWeight, BeltSpeed, RejectFlap and the rest new and ticked, the SkuRecipe structure and the LastTenWeights array unable to be imported with the reason beside them.

3. Untick what you do not want (**Tick all new** and **Untick all** help), then **Add to the connection**.
4. The rows appear in the **Signals** grid as unsaved edits, and **Why** is filled with *Signals imported from plc-tags.xlsx* unless you had written something. Check them, then **Save and activate**. Nothing reaches the node before that: the import is a faster way to type rows, and the save validates and records them as one revision like any other.

Tag drift

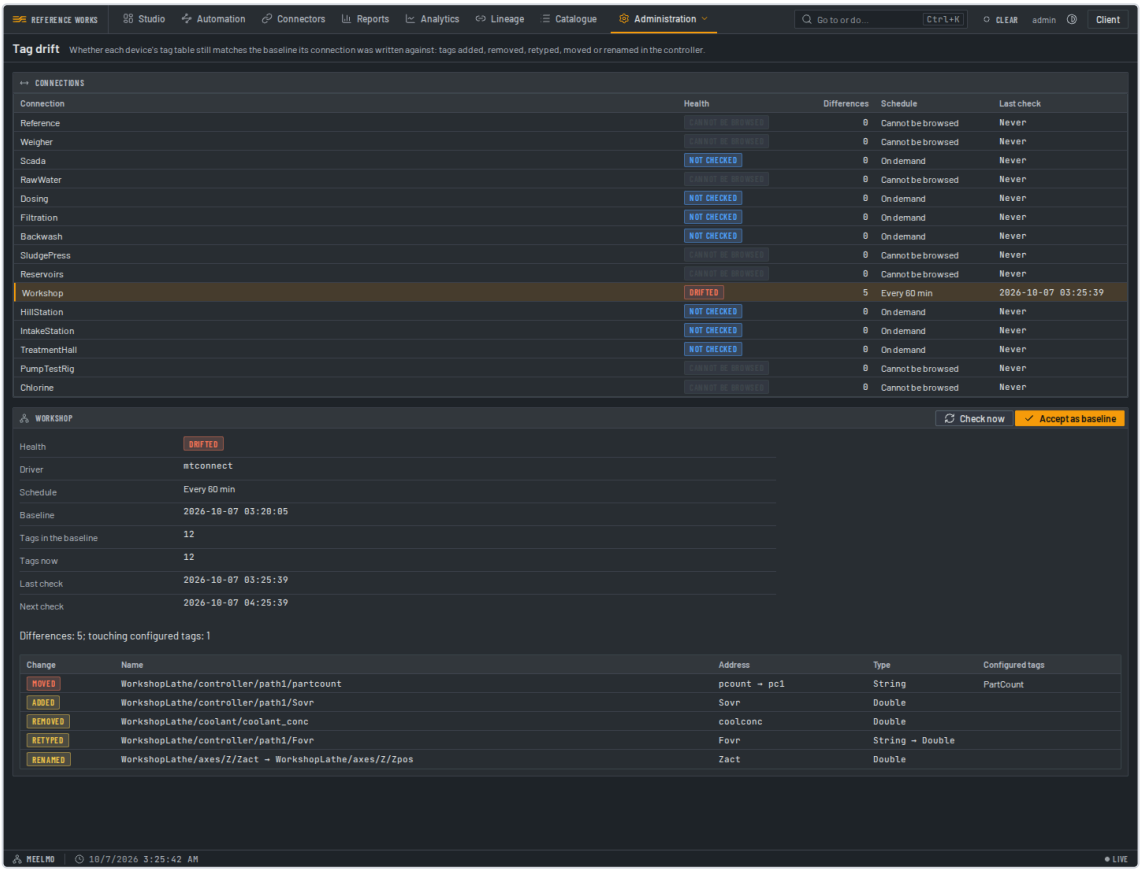
The **Tag drift** page, among the administration screens — offered to whoever the node lets change its I/O, an engineer as well as an administrator — lists every connection with how its device's tag table stands against its baseline — **Matches**, **Drifted**, **Browse failing**, **Not checked** or **Cannot be browsed** — the number of differences, the schedule and the last check. Select a connection for the baseline (when, and who accepted it), the latest browse, and each difference: **Added**, **Removed**, **Retyped**, **Moved** or **Renamed**, with the configured signals it touches.

- **Check now** browses the device at once.
- **Accept as baseline** asks first, then makes the latest browse the baseline; the node keeps who accepted it, and the data API's audit records `api.drift.accept`.

The page works the same on a plant server, where the recorder service holds the node and this web host serves it: the host reads the node's drift monitor, and **Check now** and **Accept as baseline** are done at the node as the person signed in, so the node checks their permission to change I/O and its trail names

them (`drift.check` , `drift.accept`). There the page is open to anybody the node lets change I/O, not only to administrators: every browser on a plant server reads with one shared account that may change nothing, so what a person may do is what the node granted them.

What is checked, how gently, and on which editions is in *Acquisition and I/O configuration ▸ Tag drift*.



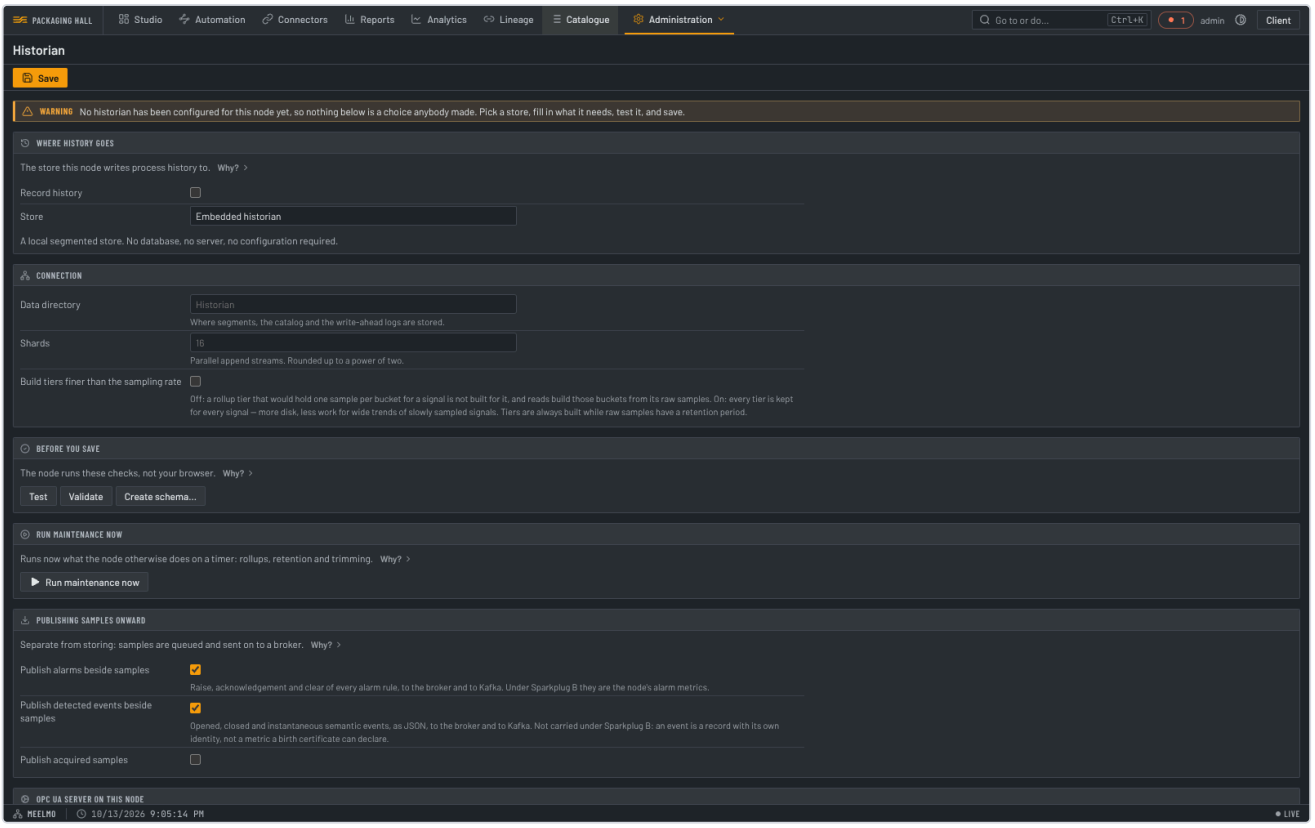
Tag drift in the browser: every connection of the reference works with its state, the Workshop lathe Drifted with five differences and checked every 60 minutes; below, its baseline and latest browse, and the five differences — moved (touching PartCount), added, removed, retyped and renamed

Lineage

Lineage moved out of administration: it is in the top bar for everybody. It is described in *Reports and analysis in the browser ▸ Lineage*.

Historian

Administration ▸ Historian chooses where this node writes process history, tests it, and runs maintenance. It is offered to whoever the node lets configure its historian (an engineer as well as an administrator); building a schema also needs the node's permission to provision one, which the node asks when it is done. A new node shows a warning — "No historian has been configured for this node yet, so nothing below is a choice anybody made. Pick a store, fill in what it needs, test it, and save."



The Historian screen on a node with nothing configured: Record history unticked, the Embedded historian chosen as store, its data directory and shards, the Test, Validate and Create schema buttons, Run maintenance now, and Publish acquired samples.

Save (top) stores the choice as a configuration revision at the node — "Saved. History will go to ...". The node opens the store itself; if it cannot, it goes on writing to the one it had, keeps trying, and says why in its log.

Where history goes

Field	Meaning
Record history	whether this node records at all
Store	**Embedded historian** (a local segmented store; no database, no server, no configuration required) or a database store — **SQLite**, and **Microsoft SQL Server**, **TimescaleDB**, **QuestDB**, **ClickHouse** or **Oracle Database** where the corresponding historian module is licensed. A store this host cannot offer is marked "not available here"
Copy existing history	appears, ticked, when the save would move history to another store and the store in use now holds history, on a host that runs the node itself. The node switches first and then copies what the old store held before the switch into the new one, in the background, at most 50,000 samples a second; the old store is only read and is kept

Copying history

While a copy runs, and after it stops, the **Copying history** panel shows **State** (Copying, Finished, Stopped or Cancelled), **Copying** (from which store into which, up to when), **Progress**, **Samples copied** with the rate and its ceiling, **Time left**, and anything **Not copied**. **Cancel copy** stops it and keeps what was copied; a stopped copy says why and offers **Retry**, which carries on from where it was, and **Dismiss**. The panel follows the copy every two seconds while it runs. *The historian*, *Copying history when you switch stores*, says what is copied and why the old store is not read alongside meanwhile.

Connection

The fields the chosen store needs. For the **Embedded historian**:

Field	Meaning	Default
Data directory	where segments, the catalogue and the write-ahead logs are stored	`Historian`
Shards	parallel append streams, rounded up to a power of two	`16`
Build tiers finer than the sampling rate	off: a rollup tier that would hold one sample per bucket is not built for that signal, and reads build those buckets from the raw samples. On: every tier is kept for every signal — more disk, less work for wide trends of slowly sampled signals	off

A database store asks for its server, database, account and password instead; a stored password shows "unchanged — type to replace".

Before you save

"The node runs these checks, not your browser." — it has the providers, the credentials and the network the historian will be used over.

Button	Does
Test	reaches the database
Validate	checks the schema and the account's rights; "The database is reachable and the tables are not there yet. Create schema builds them."
Create schema…	issues DDL to create or upgrade the tables, after a confirmation. It is idempotent and is not undone by rolling back a revision

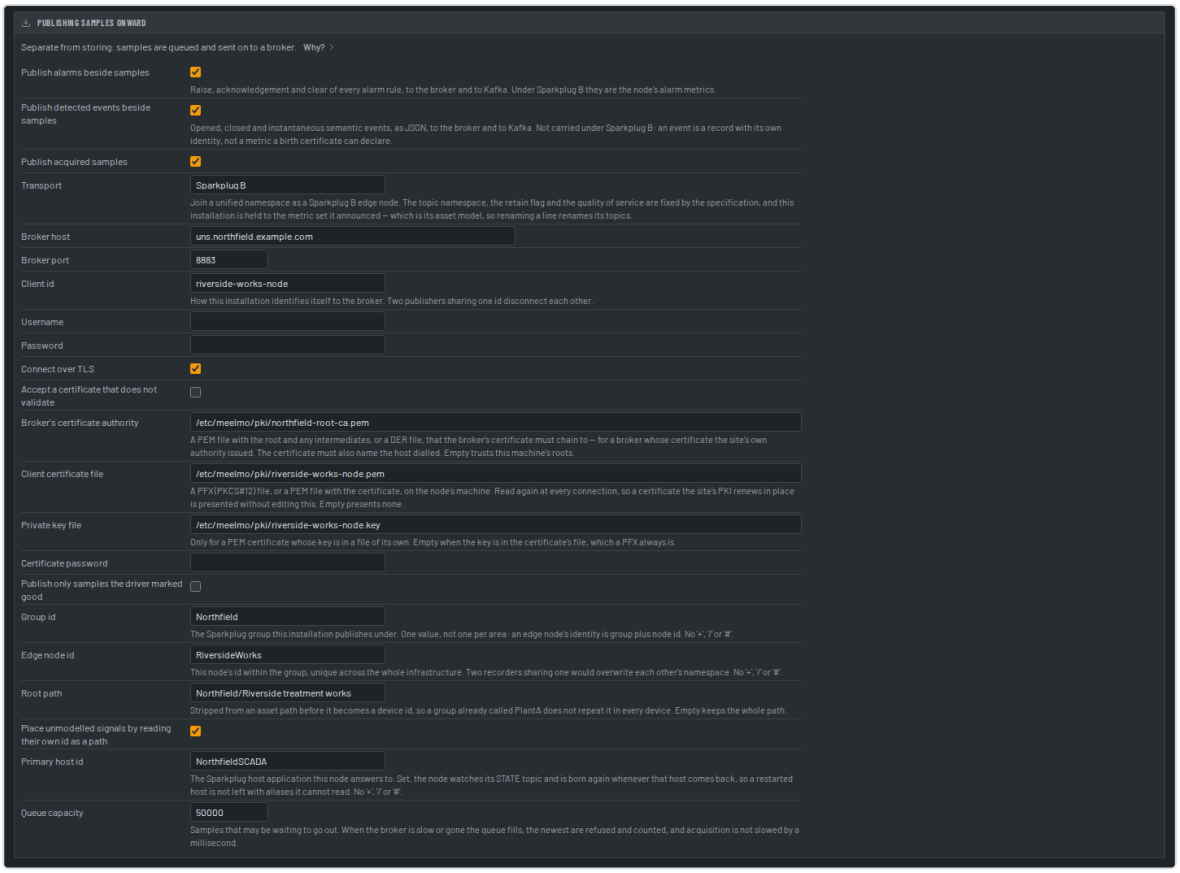
Run maintenance now

Run maintenance now runs immediately what the node otherwise does on a timer: builds rollup buckets, applies retention and trims the semantic model. The timer waits a full interval before its first pass, so on a fresh installation there is nothing to see for an hour without this button.

Publishing samples onward

The node's own OPC UA server, the Kafka producer and publishing over a client certificate are panels of this screen too; *Industrial integration* shows each.

Separate from storing: acquired samples are queued and sent to a broker by a background pump, so a slow or absent broker never delays acquisition. It needs the Industrial integration module. **Publish acquired samples** turns it on and shows its fields: **Transport** (MQTT or Sparkplug B), **Broker host**, **Broker port**, **Connect over TLS**, **Accept a certificate that does not validate** (with a warning), **Username**, **Password** (with **Forget the stored password**), **Client id**; for MQTT (one JSON message per sample) **Quality of service** (0, 1 or 2), **Ask the broker to keep the last value per topic** and **Topic prefix**; for Sparkplug B (an edge node in a unified namespace, whose topics, retain flag and quality of service the specification fixes) **Group id** and **Edge node id**; then **Root path**, **Place unmodelled signals by reading their own id as a path**, **Publish only samples the driver marked good** and **Queue capacity**.



Publishing samples onward on the reference works: Sparkplug B to uns.northfield.example.com on port 8883 over TLS, the broker's certificate authority, the node's client certificate and private key from /etc/meelmo/pki, group Northfield, edge node RiversideWorks, root path Northfield/Riverside treatment works and primary host NorthfieldSCADA, with alarms and detected events published beside the samples

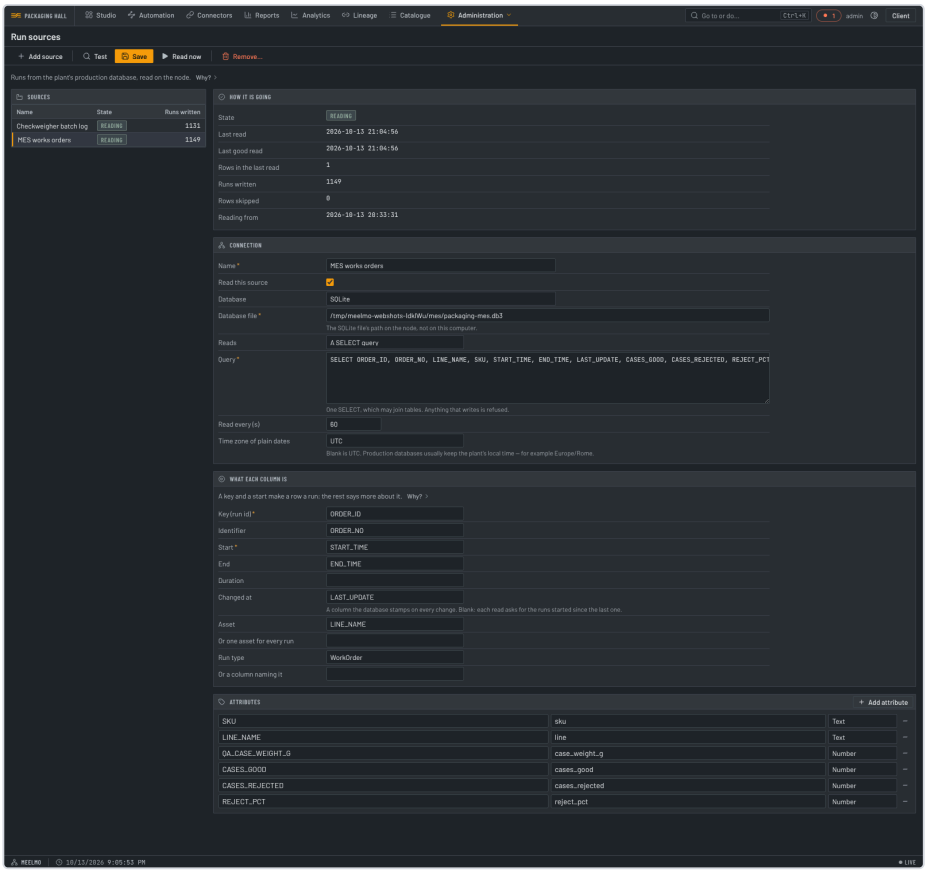
Example: turn on the embedded historian. Tick **Record history**, keep **Store** *Embedded historian*, **Data directory** `Historian`, **Shards** `16`, **Save**. After an hour of acquisition (or after **Run maintenance now**), trends over the day draw from rollups.

Run sources

Administration ▸ Run sources points the node at a plant's production database — the MES or production system that already writes a row per order, batch or lot — and brings those rows into the semantic model as **runs** with **attributes**. Every analysis over runs then has them: run summaries, process capability, run comparison and reports, with the plant's own numbers (the QA lab's case weight, the cases rejected) kept as numbers.

"Runs from the plant's production database, read on the node." The node reads each source on its own interval, from the machine that holds the node, and writes the runs into the semantic model that node keeps — the same model its detection rules write into. A node with no historian keeps no semantic model; the screen then says "This node keeps no semantic model, so nothing read here can be written" and each source shows **Nowhere to write**.

The sources are listed on the left with their state and how many runs each has written. **Add source** starts a new one; picking one in the list opens it on the right.



Run sources on the packaging node: MES works orders and Checkweigher batch log listed as Reading with the runs each has written, and MES works orders open — its connection to the MES database as SQLite, its SELECT, what each column is (ORDER_ID the key, ORDER_NO the identifier, START_TIME and END_TIME, LAST_UPDATE as changed at, LINE_NAME the asset, run type WorkOrder), its attributes kept as text or number, and how it is going.

State	Meaning
Not read yet	saved, and the node has not reached its first read
Reading	the last read succeeded
Failing	the last read failed; **How it is going** says why, and the node keeps trying on the interval
Switched off	**Read this source** is unticked
Nowhere to write	the node has no historian, so no semantic model
Not in this edition	the database kind needs a driver this edition does not carry

Connection

Field	Meaning
Name	what the source is called here and in the audit trail
Read this source	whether the node reads it; a source can be kept switched off
Database	**SQLite** in every edition; **Microsoft SQL Server**, **Oracle Database** and **PostgreSQL** where the edition carries the driver. The Community Edition carries SQLite only; the others are listed and marked "not in this edition"
Database file (SQLite)	the file's path **on the node**, not on the computer running the browser. It is opened read-only

Field	Meaning
Connection string (other databases)	an ADO.NET connection string without its password. A password pasted inside it is lifted out and sealed like the one below
Password	sealed at the node and never sent back; a stored one shows "unchanged — type to replace"
Reads	**A table or view**, or **A SELECT query** (which may join tables)
Table / **Query**	what to read. A query must be one SELECT (or a WITH that ends in one); anything that writes, locks or runs code — `INSERT`, `UPDATE`, `DELETE`, `SELECT … INTO`, `FOR UPDATE`, a second statement after a semicolon — is refused before it is sent
Read every (s)	the interval, from 5 seconds to once a day
Time zone of plain dates	the zone of date columns that carry no offset; blank is UTC. Production databases usually keep the plant's local time, so set it (for example `Europe/Rome`)

Use an account that can only read. The node sends nothing but a SELECT, but a server database has no client-side switch that makes a read-write account read-only.

What each column is

"A key and a start make a row a run; the rest says more about it."

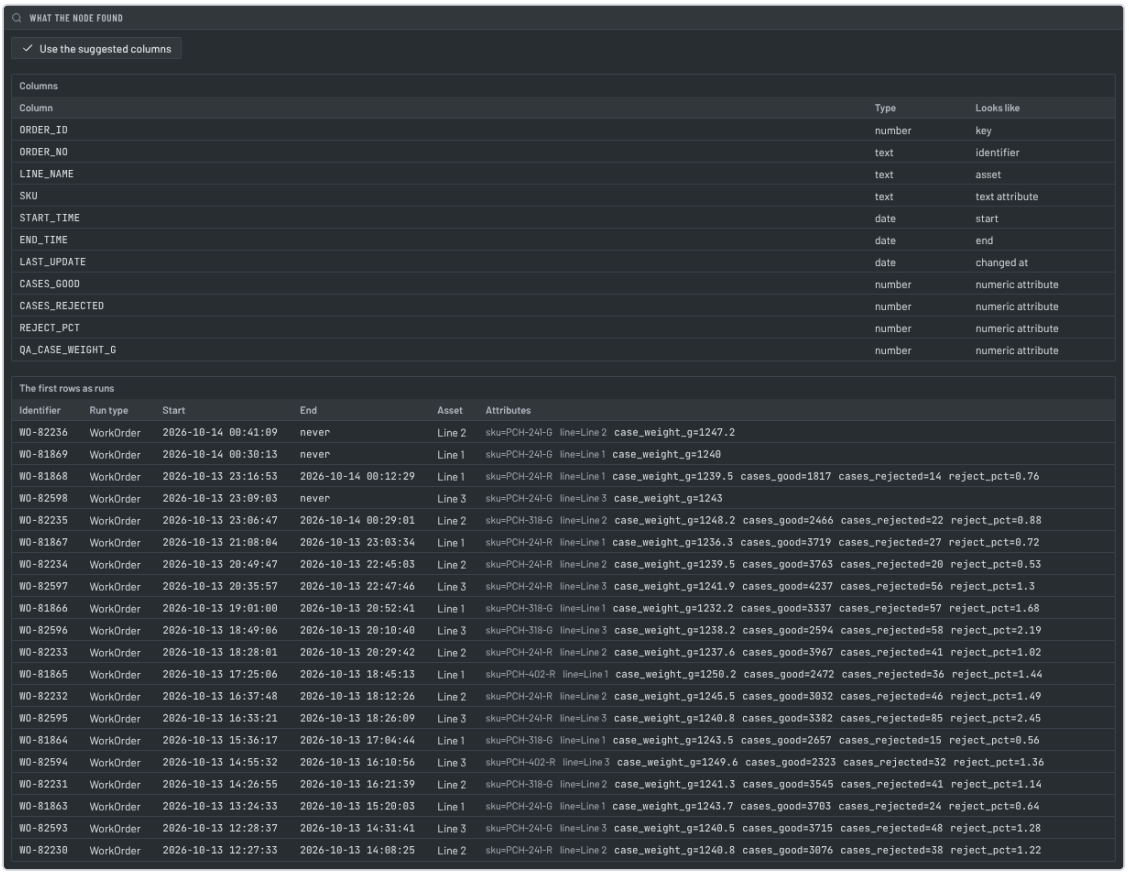
Field	Meaning
Key (run id)	the column that makes a row the same run the next time it is read. The run's id is the source's id and this value, so reading a row again updates its run instead of adding another
Identifier	what the plant calls the run — a works order number. Blank uses the key
Start	when the run started. Required
End / **Duration**	when it ended, or how long it took (**Duration in** seconds, minutes or hours). A row with neither is a run still going
Changed at	a column the database stamps on every change, such as `LAST_UPDATE`. With one, each read asks for the rows changed since the last; without one, for the rows started since — and a run read while it was still open holds the next read back to its start (for at most two days) so its end is picked up when it comes
Asset / **Or one asset for every run**	the column naming the line, machine or area each run belongs to, or one asset for all of them
Run type / **Or a column naming it**	what kind of run these are — `WorkOrder` — or a column naming each row's kind. Blank uses the source's name

Attributes lists the columns carried onto every run: the **Column**, its **Name on the run** (blank keeps the column's name) and whether it is **Kept as Text** or **Number**. A number is kept as a number whatever the database called it — `REAL`, `NUMBER(10,2)`, or text that holds a number — so analyses can compute with it. A value that is not a number is left off that run and reported.

Testing before saving

Test (toolbar) — "The node reads the first rows now, from where it will read them. Nothing is saved or written." It shows, under **What the node found**:

- **Columns**: each column, its **Type** (text, number, date, mixed) and what it **Looks like** — key, identifier, start, end, changed at, duration, asset, run type, numeric or text attribute. The guesses follow the names and values: a unique column named like `…_ID` or `…COUNTER` is a key, `START…` / `BEGIN…` a start, `STOP…` / `END…` / `FINISH…` an end, `LAST_UPDATE` a changed column, a repeated `AREA…` / `LINE…` / `MACHINE…` an asset, and numbers that are not keys numeric attributes.
- **Use the suggested columns** fills only the blank fields; nothing already chosen is replaced. Test again to see the runs.
- **The first rows as runs**: the most recent rows as the runs they would become — identifier, type, start and end in UTC, asset, and the attributes, numeric ones in figures.
- Rows that would not become runs, or would lose a value, and why.



What the node found when MES works orders was tested: each column with its type and what it looks like — ORDER_ID a key, ORDER_NO the identifier, LINE_NAME the asset, START_TIME, END_TIME and LAST_UPDATE the start, end and changed at, the counts and the weight numeric attributes — and the first rows as works orders with their line and attributes, the open ones ending never.

Save stores the source beside the node (`run-sources.json` in the node's directory, the password sealed) and records `runsource.added` or `runsource.changed` in the audit trail under your name — the host log on **Audit trail**. **Read now** reads it at once instead of waiting for the interval. **Remove…** asks first; the node stops reading the database, and the runs it has already written stay. Each read takes at most 5,000 rows and continues from where it stopped on the next one, so a large table is brought in over several reads.

These routes are served where the node is held, and on a plant server — where the recorder service holds the node and has no screen of its own — this web host serves them by asking the node as the person signed in: the node lets only an administrator (the permission to manage users) list, test, save, read now or remove a source, tests from its own machine, seals the password against its own key, and records `runsources.save`, `runsources.sync` and `runsources.delete` in its trail under that person (and `runsource.added` or `runsource.changed` in its log). This host passes the password to the node in the request that saves it and keeps no copy. Whether to ask at all is decided by the node's grant to the

person, which this host keeps from the node's answer on every request — not by the account the browser reads with, which on a plant server is shared and may only read. A host attached to no node says run sources belong to a node, and are set up where it runs.

Example: the packaging hall's works orders. The MES keeps `WORK_ORDERS` — one row per order: `ORDER_ID`, `ORDER_NO`, `LINE_NAME`, `SKU`, `START_TIME`, `END_TIME`, `LAST_UPDATE`, `CASES_GOOD`, `CASES_REJECTED`, `REJECT_PCT` and the QA lab's `QA_CASE_WEIGHT_G` — with its dates in UTC. This is the source in the figure.

1. **Add source**. **Name** `MES works orders`, **Database** *SQLite*, **Database file** `D:\MES\packaging-mes.db3`, **Time zone of plain dates** `UTC`.
2. **Reads** *A SELECT query*: `SELECT ORDER_ID, ORDER_NO, LINE_NAME, SKU, START_TIME, END_TIME, LAST_UPDATE, CASES_GOOD, CASES_REJECTED, REJECT_PCT, QA_CASE_WEIGHT_G FROM WORK_ORDERS`
3. **Test**. The columns come back with `ORDER_ID` as key, `START_TIME` / `END_TIME` as start and end, `LAST_UPDATE` as changed at, `LINE_NAME` as asset, and the counts and the weight as numeric attributes. **Use the suggested columns**, set **Identifier** `ORDER_NO` and **Run type** `WorkOrder`, name the attributes `sku`, `line`, `case_weight_g`, `cases_good`, `cases_rejected` and `reject_pct`, and **Test** again: the first orders appear as runs.
4. **Save**, then **Read now**. **How it is going** shows **Reading** and the number of runs written. A run summary of type `WorkOrder` with those attributes as context fields — the `orders` dataset of *Reports and analysis in the browser* — now lists the orders, with the case weight and the reject share as numbers.

The checkweigher's own batch log, `CW_BATCHES`, is a second source over the same file (**Reads** *A table or view*, run type `CheckweigherBatch`), which is what the cross-check in that chapter holds the orders against.

Linked nodes

Administration ▸ Linked nodes lets this host **read** other plants beside its own, so one project can mix their tags: a trend with a trace from each line, a dataset whose columns come from two plants, an HMI value from the next building. "Other plants this host reads, beside its own. Read only."

This host still belongs to **one node** — its *home node*. People sign in there, and every write, command, setpoint, configuration change and administration screen goes there and only there. A linked node is read, and never written.

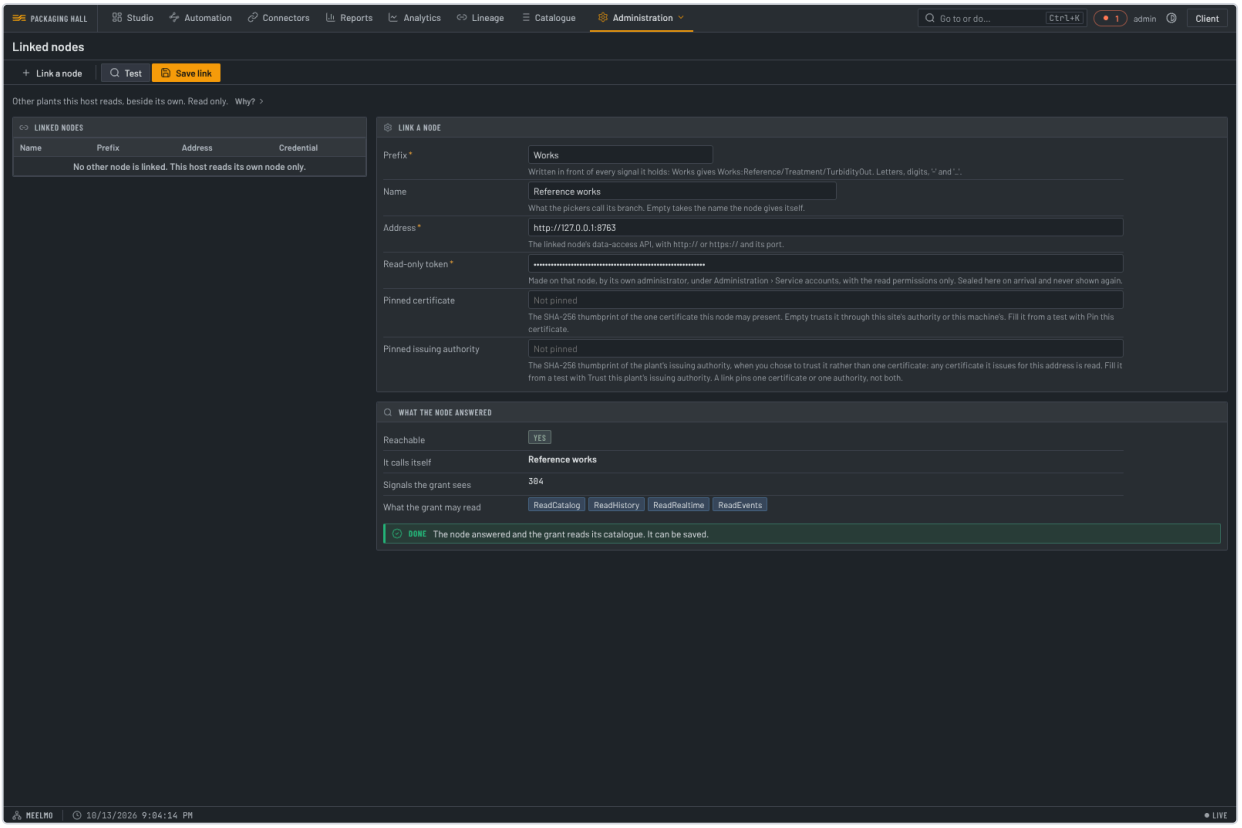
What an administrator brings

1. On the **other** node, its own administrator makes an integration account for this host under **Administration ▸ Service accounts**, with the read permissions — *Read catalogue*, *Read history*, *Read live values*, *Read events* — and **not** *Administer*. That node then decides what this host may read there (its signal scope, its range limits) and records every read in its own audit trail under that account.
2. Here, **Link a node**, and fill in:

Field	Meaning
Prefix	written in front of every signal of that node: `Works` gives `Works:Reference/Treatment/TurbidityOut`. Letters, digits, `-` and `_`. A prefix this node's own signals already begin with is refused.
Name	what the signal pickers call the node's branch; empty takes the name the node gives itself
Address	the other node's data-access API, `https://works-server:8443`
Read-only token	the token of the account made in step 1. "Sealed here on arrival and never shown again."

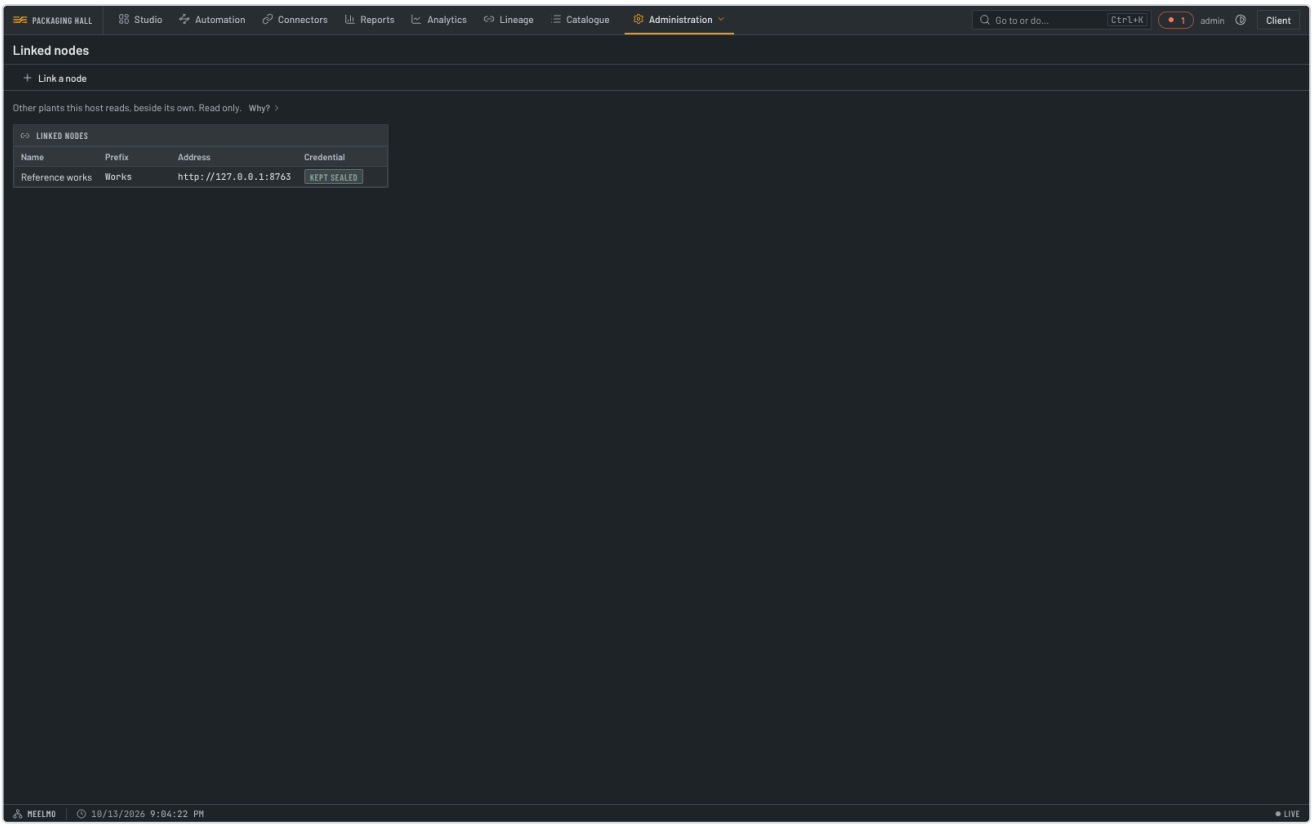
Field	Meaning
Pinned certificate	over HTTPS, the SHA-256 thumbprint of the one certificate that node may present; empty trusts it through this site's authority or this machine's. Filled from a test with **Pin this certificate**.
Pinned issuing authority	instead of one certificate, the SHA-256 thumbprint of the authority that issues that plant's certificates, when you chose to trust it (see Certificates). Filled from a test with **Trust this plant's issuing authority**. A link pins one certificate or one authority, not both.

3. **Test** asks the node with that token and saves nothing: **Reachable**, what **It calls itself**, how many **Signals the grant sees**, and **What the grant may read**. A token that may administer the other node is refused: "This token may administer that node. Make a read-only integration account there instead..." — this host never holds a key to another plant's accounts.



Link a node filled in for the reference works: prefix Works, name Reference works, its address and the read-only token as dots, no pinned certificate over plain HTTP; What the node answered below it — reachable, the name it gives itself, the signals the grant sees and the four read permissions — and a line saying the grant reads its catalogue and the link can be saved.

4. **Save link** tests again and keeps only a link that works. The token is kept sealed in this host's configuration file with its other secrets; no screen, response or trail entry ever shows it again. The list shows **Kept sealed** in its place.



Linked nodes after the link is saved: the list with Reference works, its prefix Works, its address and its credential Kept sealed.

Change opens a saved link in the same form. Its prefix cannot change; the token box starts empty, and left empty keeps the token this host holds. A change is tested before it is saved, like a new link. **Test** on a saved link asks the node with the token this host holds — what it presents and what it lets this host read now.

Remove link asks first. This host stops reading that node; nothing is changed on the other node, whose account for this host stays until its own administrator removes it.

Certificates

Over HTTPS, **Test** also shows the certificate the node presented: its **SHA-256 thumbprint**, **Issued to**, **Issued by**, **Valid until** and **Trust**:

Trust	Meaning
Trusted by this site's authority or this machine's	it chains to the authority this host pinned when it joined its own node, or to the machine's trust store, and names the address dialled. A node of the same site needs nothing more.
Not trusted without a pin	a plant with its own authority. Nothing is sent to it — not the token, not the question — until its certificate is pinned.
The pinned certificate	it is exactly the certificate pinned for this link.
Issued by the pinned authority	the link trusts that plant's issuing authority, and this certificate is one it issued for the address dialled.

Pin this certificate puts the thumbprint into the link. Before pinning, read it to the other node's administrator and check it against what their server shows. From then on this host reads that node only while it presents exactly that certificate — even another certificate from the same authority is refused.

When the other plant renews its certificate, the link stops being read: its share of every screen is missing and the line across the top says "Reference works presented a certificate this host does not trust, so nothing was read from it. An administrator checks its certificate and pins it under Administration › Linked nodes…". Pick the link, **Test**, check the new thumbprint with that plant's administrator, and **Pin this certificate**: the link is saved with the new pin and the token it already holds.

Trusting the plant's issuing authority instead. When the node sends the authority that issued its certificate with it — a Meelmo node does — the test also shows that **Issuing authority** and its SHA-256 thumbprint, and offers a second button, **Trust this plant's issuing authority**. Pinning one certificate stays the default; this is a choice, and the page says what it trusts before you press it: "This host then reads the node while it presents any certificate that authority issues for works-server — every renewal, and any other certificate it ever issues for that name — with no new pin. A certificate it issues for another name is still refused. The other plant, not this one, then decides which certificate is its node's." A renewal under that authority is then read without anybody doing anything; a certificate from another authority — even one that calls itself by the same name — or one the pinned authority issued for another name is refused before the token is sent, and the link is reported as not trusted until it is pinned again. The **Trust** line reads **Issued by the pinned authority**. Choose it only after that plant's administrator confirms the authority is their own and reads you its thumbprint; clear the box and pin a certificate to go back.

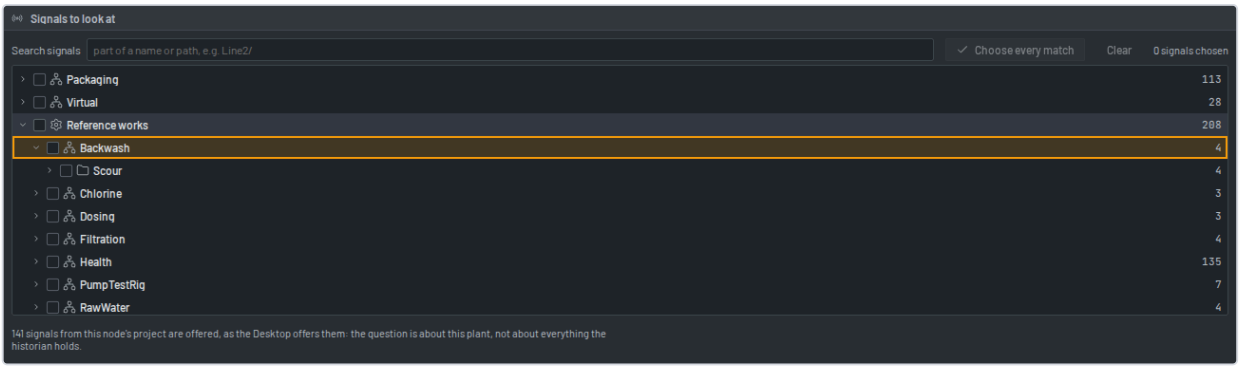
What is recorded

Adding, changing, testing and removing a link are each recorded twice, under the administrator who did it: in this host's own trail, and in the **home node's** audit trail — the one **Administration ▸ Audit** shows as the node's, where the node's other administrative changes are. The entries are `host.linked_node.added`, `host.linked_node.changed`, `host.linked_node.tested` and `host.linked_node.removed`, with the link's prefix, its address, what the grant was found to read and the certificate's thumbprint; a link that was refused is recorded with the result *refused*. The token is never in either trail. Each entry also says how the link trusts the other plant: `trust=certificate:<thumbprint>`, `trust=authority:<thumbprint> (any certificate that authority issues for this address)` or `trust=site-or-machine`, so choosing to trust a plant's authority is on record under the person who chose it.

An administrator can also change links with an API token — an integration account that holds **Administer**, as an installer's script uses — instead of signing in. The node then records the entry as that account (`svc:<id>`), which it reads from the token itself, and the entry says `through=web-host by=api-token account=<id>`. The node records these only for someone who may manage its accounts; when it cannot be asked — it is not answering, say — the change is still made and this host's entry says the node's was not written.

What people see

- The signal pickers — the Studio's **Signals** rail, the analysis pages' signal choice, the Models picker — show each linked node as a top-level branch with its name, and that node's own tree beneath. Its signals carry the prefix: `Works:Reference/Treatment/TurbidityOut`. The home node's signals keep their ids, so every existing screen, trend, dataset and report is unchanged. The analysis pages offer this node's project signals and each linked plant's, and the note under the picker counts this node's own.



What changed on the packaging node with the reference works linked: the Signals to look at picker with Packaging and Virtual, this node's own branches, and Reference works as a branch of its own with its 208 signals, opened onto the works' own connections — Backwash, Chlorine, Dosing, Filtration and the rest; the note under the tree still counts this node's 141.

- Whether a person may see linked data at all is the **home** node's decision: somebody who may not read history here reads no plant's history through this host, and an account the home node scopes to one line sees another plant's signals only if its scope includes that plant's prefix.
- When a linked node cannot be reached or refuses, its share of a screen is **missing, not empty** — its traces are gaps — and a line across the top says so: "Reference works could not be reached. Its signals are shown as missing, not as quiet; everything else on this screen is live." **Set aside** hides it until the next read that asks that node.
- A write aimed at a linked signal is refused by this host: "This signal belongs to Reference works, which this host only reads. Writes go to the signal's own node: command it from Reference works's own screens."
- **Analytics ▸ Compare places ▸ Plants** sets this plant beside each linked plant: each studies its own saved dataset of the same name against the limits stored on it, read through its own API with this host's account (see *Compare places*). A plant whose window held fewer than two items has no capability figures, but its row still shows the limit range stored on that plant and where it came from, read from that plant's own stored limits with the same read-only account: "Too few items for capability figures; the range is the limit stored on this plant."
- **Diagnostics** (`GET /api/v1/diagnostics` , for an administrator) lists the live subscriptions at this node as it always did, and beside them `linkedStreams` : for each linked node, how many streams this host holds open to it now, the frames it has sent, and how many of its streams ended in a gap.
- The alarm drawer lists every linked plant's standing alarms beside this plant's, with their prefix. Their **ACK** and **Shelve** are disabled: "This alarm belongs to Reference works, which this host only reads. Writes go to the signal's own node: answer it on Reference works's own screens." A plant that does not answer is named in the line across the top; its alarms are missing, not cleared.
- Lists of runs, events and states stay this plant's own unless they name a linked asset. That keeps every existing count unchanged when a link is added. A plant-wide answer across linked plants is asked for explicitly: `"includeLinked": true` on the data-access API's semantic queries (`POST /api/v1/semantic/executions` , `/events` , `/states`) and on the Assistant's `semantic_executions` , `semantic_events` and `semantic_states` tools. Records from a linked plant come back with its prefix, and how each plant answered is in the `Meelmo-Linked-Nodes` header — for the Assistant, in a `nodes` list in the answer. The answer is one list in start order, and `maxResults` applies to the whole of it — the earliest, or with `"latest": true` the latest, across every plant — not to each plant. A plant some of whose records were cut to make room is still *answered*, with a message such as "Reference works sent 3 records and 1 of them are in this answer: it holds the 4 earliest across every plant asked. Narrow the window or name its assets to see the rest."

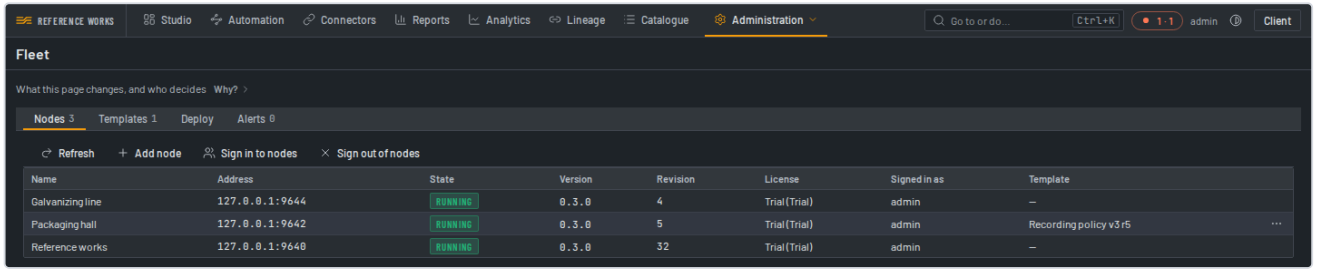
- A live stream that mixes plants is one stream: when a linked plant goes away in the middle of it, the stream carries a gap that names the plant and goes on with the others.
- This host reads a linked node as a Meelmo host, not as the .NET SDK: a plant that has not licensed the SDK does not refuse it.

Example. On the reference works' host, make a service account `Packaging hall web host` with the four read permissions and copy its token. On the packaging hall's host, **Link a node** with prefix `Works`, name `Reference works`, the works' address (`https://works-server:8443` on a site; the figures above link a host on the same machine over plain HTTP) and the token, **Test** (Reachable, the name the works gives itself, its signals, four read permissions), **Save link**. A trend on the packaging hall can now carry `Packaging/Line/ElectricalPower` and `Works:Virtual/Computed/PumpingPower` on one time axis, and **Compare plants** sets the two plants' power side by side.

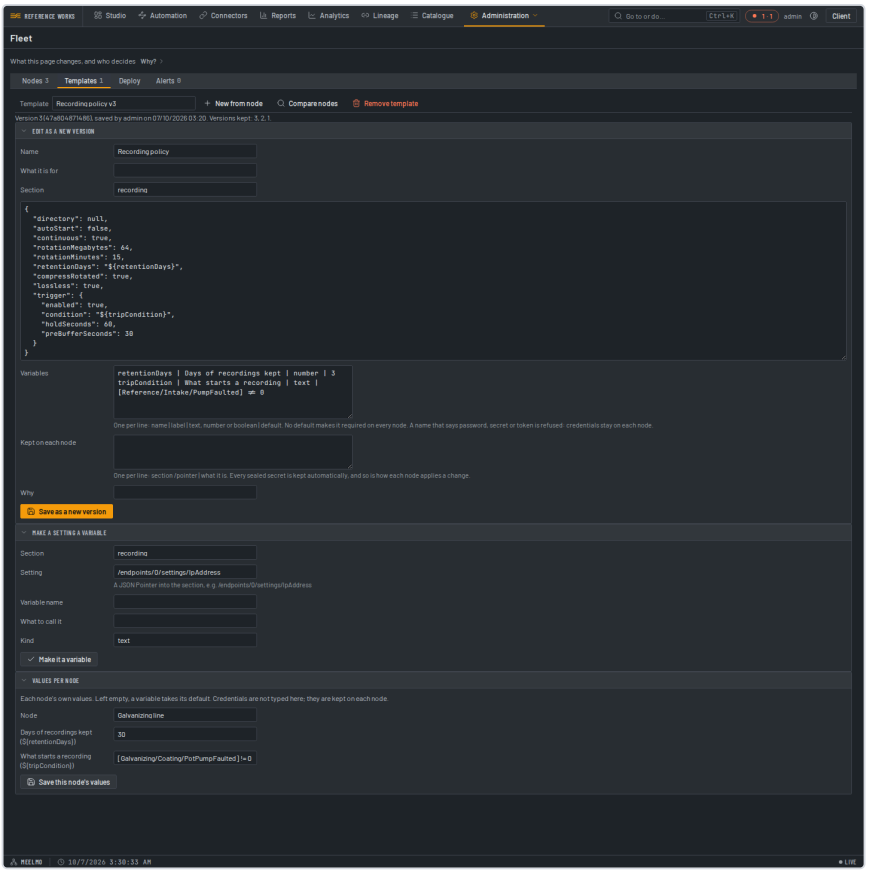
Fleet

Administration ▸ Fleet is the fleet management of the Desktop's *Fleet* and *Templates* screens, in the browser — with the two things only an unattended host can do: a rollout that goes on soaking after the page is closed, and fleet alerts at three in the morning. It needs **Administer** here; changing a node needs what that node grants you there. It is part of the Professional edition; a Community build shows that it is not included.

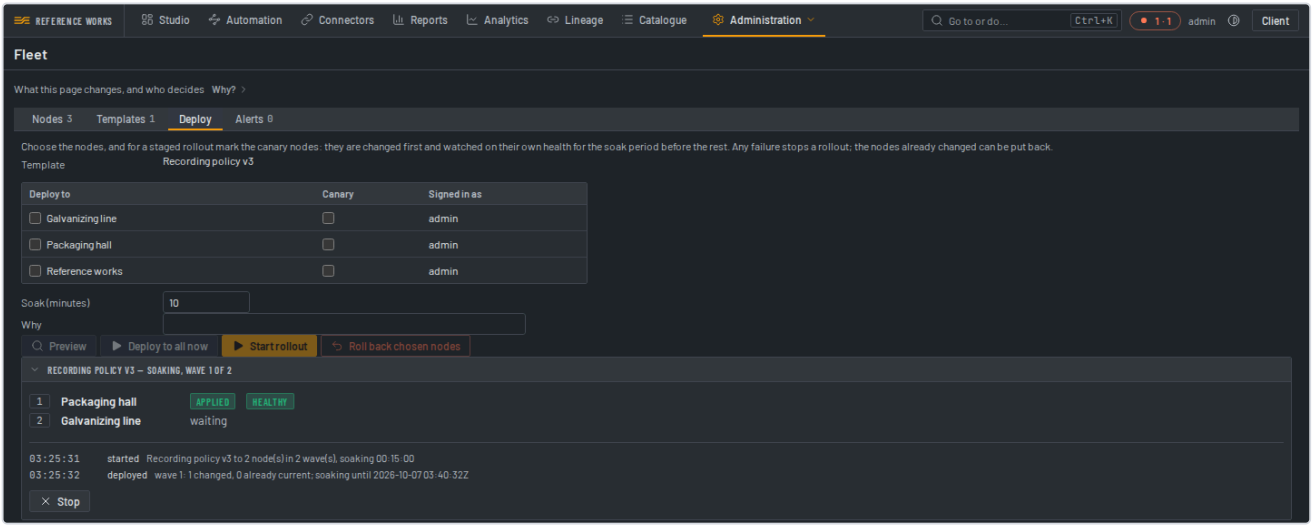
Tab	What it does
Nodes	The roster: each node's state, version (flagged when **behind** the newest in the fleet), revision, licence once you are signed in to it, and the template version it runs. **Add node**, **Sign in to nodes** (one account sent to each chosen node, which decides on its own; the password is not kept — each node's session is kept by this host for you, sealed, until the node expires it, **Sign out of nodes**, or you sign out of this host), and per node **Upgrade instructions** and **Remove from the fleet**
Templates	**New from node**, **Edit as a new version**, **Make a setting a variable**, **Values per node**, **Compare nodes** (in sync, or edited at the node, and where) — as on the Desktop
Deploy	**Preview**, **Deploy to all now**, **Start rollout** (canary nodes, a soak in minutes, then the rest), **Stop**, **Roll back**, with every node's result on its own line and each rollout's timeline
Alerts	The fleet alert rules and where they go



Administration ▸ Fleet, Nodes tab: the galvanizing line, the packaging hall and the reference works, each running, with its version, revision, licence, who is signed in there and the template it runs



The Templates tab with Recording policy: the recording section with ${retentionDays} and ${tripCondition} in it, the two variables, Make a setting a variable, and the galvanizing line's own values



The Deploy tab with a staged rollout of Recording policy soaking: the packaging hall applied and healthy as the canary, the galvanizing line waiting for the soak to end, and the timeline

Templates

A template is a set of configuration sections taken from one node and carried to others, with the settings that differ per node made into variables.

Control	Does
Template	Chooses the template to work on. The line under the toolbar says "Version N (hash), saved by … on …. Versions kept: …"

Control	Does
New from node	Opens a panel: **From which signed-in node**, a **Name**, and a tick box per section to carry (`acquisition`, `analysis`, `alarms`, `automation`, `recording`, `semantic`, `hmi`, `definitions`, `historian`). **Make the template** reads them from that node
Compare nodes	Lists every node with its **State** against the template — **in sync**, **edited at the node**, **cannot be compared** or **could not be read** — and, under **Where**, what differs
Remove template	Asks first: "Remove the template … and all its versions? Nodes keep running what they were given."

Edit as a new version (folds open):

Field	Meaning
Name, **What it is for**	The template's name and description
Section	Which carried section the text box below shows, as JSON, for editing
Variables	One per line: `name | label | text, number or boolean | default`. No default makes the variable required on every node. A name that says password, secret or token is refused: credentials stay on each node
Kept on each node	One per line: `section /pointer | what it is`. Settings each node keeps as its own. Every sealed secret is kept automatically, and so is how each node applies a change
Why	The comment saved with the version
Save as a new version	Saves it: "Saved … as version N." Earlier versions are kept

Make a setting a variable (folds open) turns one setting into a variable without editing the JSON by hand:

Field	Meaning
Section	One of the template's sections
Setting	A JSON Pointer into the section, for example `/endpoints/0/settings/IpAddress`
Variable name	The name used in the template, written `${name}`
What to call it	The label shown under **Values per node**
Kind	`text`, `number` or `boolean`
Make it a variable	Puts `${name}` in place of the setting and saves the template as a new version: "Saved … as version N."

Values per node (folds open) holds each node's own values. Choose the **Node**; a box per variable follows, labelled with the variable's label and `${name}`, and showing its default or *required* when empty. "Left empty, a variable takes its default. Credentials are not typed here; they are kept on each node." **Save this node's values** stores them.

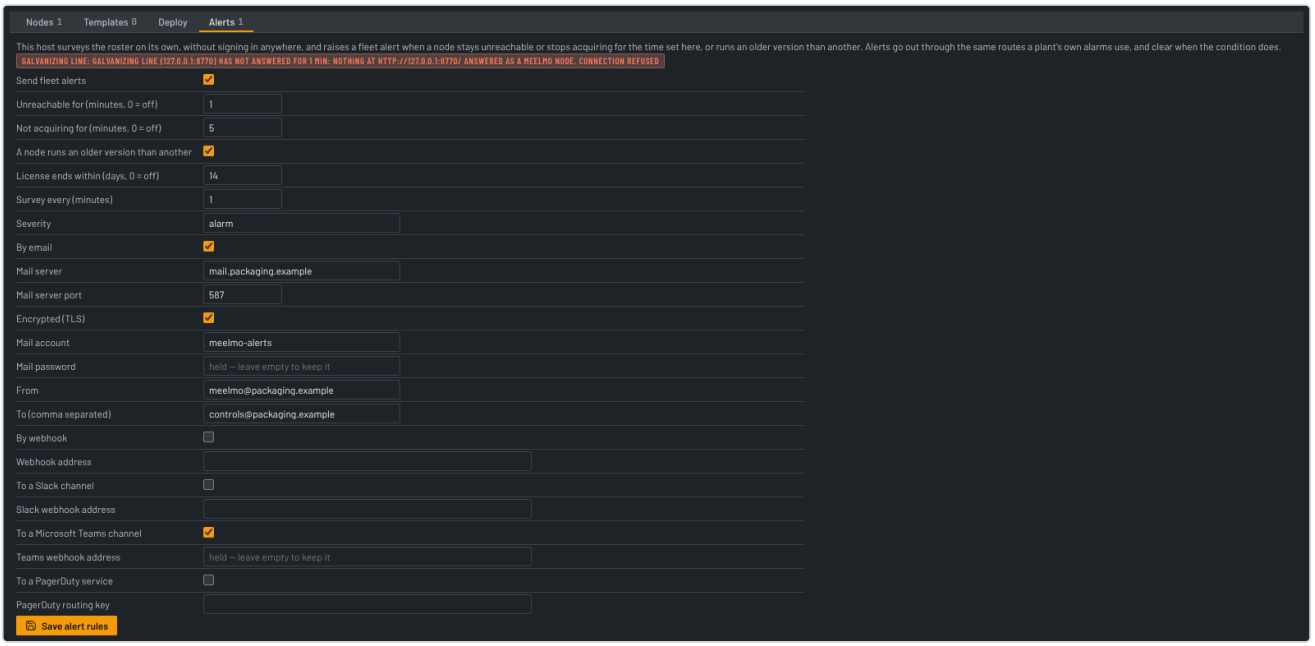
Example: one connection template for two packaging lines. On a template made **New from node** from the packaging hall with the `acquisition` section ticked, open **Make a setting a variable**: **Section** `acquisition`, **Setting** `/endpoints/0/settings/IpAddress`, **Variable name** `plcAddress`, **What to call it** `PLC address`, **Kind** `text`, then **Make it a variable**. Under **Values per node**, choose each node in turn, type its own PLC address in **PLC address (${plcAddress})**, and **Save this node's values**. **Compare nodes** then shows which nodes are **in sync** before you deploy.

Fleet alerts

Alerts sets the fleet alert rules and where they go. Every field:

Field	Meaning
Send fleet alerts	Switches the survey's alerts on
Unreachable for (minutes, 0 = off)	Raise an alert when a node has not answered for this long
Not acquiring for (minutes, 0 = off)	Raise an alert when a node has stopped acquiring for this long
A node runs an older version than another	Raise an alert when a node is behind the newest version in the fleet
License ends within (days, 0 = off)	Raise an alert this many days before a node's licence ends. Only for nodes where a session could read the licence
Survey every (minutes)	How often the host surveys the roster: 1 to 60
Severity	The severity the alerts are raised with: **information**, **warning**, **alarm** or **critical**
By email	Send alerts by email, with **Mail server**, **Mail server port** (1–65535), **Encrypted (TLS)**, **Mail account**, **Mail password**, **From** and **To (comma separated)**
By webhook	Post alerts to **Webhook address**
To a Slack channel	Post alerts to **Slack webhook address**
To a Microsoft Teams channel	Post alerts to **Teams webhook address**
To a PagerDuty service	Send alerts with **PagerDuty routing key**
Save alert rules	Stores the rules: "The alert rules were saved."

The **Alerts** tab shows how many alerts are active, and lists each above the fields. A secret already stored (the mail password, the Slack and Teams addresses, the PagerDuty key) shows *held — leave empty to keep it*.



The Alerts tab with one alert standing: the galvanizing line has not answered for a minute; below it the rules — unreachable for 1 minute, not acquiring for 5, an older version, the licence within 14 days, a survey every minute, severity alarm — then by email through mail.packaging.example with the mail password held, and to a Microsoft Teams channel with its address held.

This host surveys the roster on its own, without signing in anywhere — whether a node answers, its version and whether it is acquiring are things every node tells anybody — and raises an alert when a node stays **unreachable** or **not acquiring** for the minutes set here, **runs an older version** than another node, or (where a session could read it) its **licence ends** within the days set here. An alert is delivered through the same routes a plant's own alarms use — email, webhook, Slack, Microsoft Teams, PagerDuty — and clears when its condition does. Every secret typed here is sealed by this host's secret protector on arrival and is never shown again; an empty box keeps it. Every raise and clear, and every change on this page, is in this host's audit trail.

Across a restart, and where it is kept

Node sessions across a restart. The session each node issued you is kept per person in `fleet/fleet-sessions.json`, sealed by this host's secret protector — the same AES-256-GCM sealing, under the same host-bound master key, as the alert secrets — so restarting the host does not sign anybody out of the nodes. Each session is sealed whole (whose it is, which node, the token, when it expires), so nothing in the file can be read, edited or moved to another person. The master key is wrapped by the machine on Windows and readable only by the service account elsewhere, so the file copied to another machine opens nothing; where the machine has a stable identity of its own (not a container), a session is also refused on any other machine. A session that cannot be opened is dropped and logged, and its person signs in to that node again. A kept session expires when the node's does — the node's idle window, moved on each time it is used — and is removed when you choose **Sign out of nodes**, when you sign out of this host, and when the node is removed from the fleet.

A rollout across a restart. A rollout that was deploying, soaking or rolling back when the host stopped is carried on when it starts again, as the person who was running it:

- a wave being deployed is deployed again, and a node already running this rollout's revision (its revision comment names the rollout) is recognised rather than changed twice;
- a soak picks up with the time it had left when the host stopped — the time nobody was watching does not count as healthy;
- a rollback carries on with the nodes not yet put back, and a node already put back is reported as such, not rolled back twice.

If that person's session on a node the rollout still has to call did not survive — it expired at the node, or they signed out — the rollout **halts** instead, and its timeline says whose session is missing and on which nodes. Sign in to them, then **Roll back** or start the rollout again. Stopping the host is never recorded as somebody's **Stop**.

Where it is kept. `fleet/fleet-roster.json`, `fleet/fleet-library.json`, `fleet/fleet-alerts.json` and `fleet/fleet-sessions.json` in the host's state directory — addresses, templates, node values, rollouts, sealed alert secrets and sealed node sessions.

Single sign-on

Administration ▸ Single sign-on lets people sign in through the site's OpenID Connect provider beside this installation's own accounts, never instead of them: local accounts keep working on the morning the provider cannot be reached. The sign-in is the authorization code flow with PKCE. It has been proven end to end against Keycloak; Entra ID and Okta are not yet proven. Identity stays per node: nothing here reaches another node.

The settings are kept in this host's `web-config.json` and read when the host starts. Two banners under the toolbar say where you stand:

- *This host started with sign-in through the provider, and offers it on the sign-in card.* or *This host started without sign-in through a provider. Saved settings are used from the next start.*
- After **Save**: *Saved to web-config.json. The host reads this when it starts: restart the browser host for sign-in to use it.*

A host with no configuration file to write says so, and **Save** is greyed. A client secret the file holds in the clear, or one sealed on another machine, is named in a warning that asks you to type it again here.

Field (panel Provider)	Meaning
Offer sign-in through the provider	Ticked, the sign-in card offers the provider's button
Authority (issuer) (required)	HTTPS only. The issuer the provider's discovery document names, exactly — for Entra ID the tenant's own address, never `/common`
Client id (required)	What this host was registered as at the provider
Client secret	Sealed by the host against this machine's key, as `--seal-secret` does, and never shown again. Empty for a public client that relies on PKCE alone. A stored secret shows as kept; leave the box empty to keep it
Forget the stored secret	Only when a secret is stored. Ticked, **Save** removes it
Scopes	Separated by spaces. `openid` is always sent
Account name claim	Names the person for the session and the trail: `upn` or `email` where the provider puts the name there
Groups claim	The claim the provider's groups arrive in
Button label	What the sign-in card's button says — "Sign in with Contoso". Empty uses the card's own words in the reader's language
Redirect URI to register	Read only. The address to register at the provider: under the host's public address when it has one, otherwise under the address this page was opened at

Groups and what they grant maps the provider's groups to what their members get here, exactly as an LDAP directory's groups are mapped. **Add mapping** adds a row; **Remove this mapping**, the button at its end, removes it.

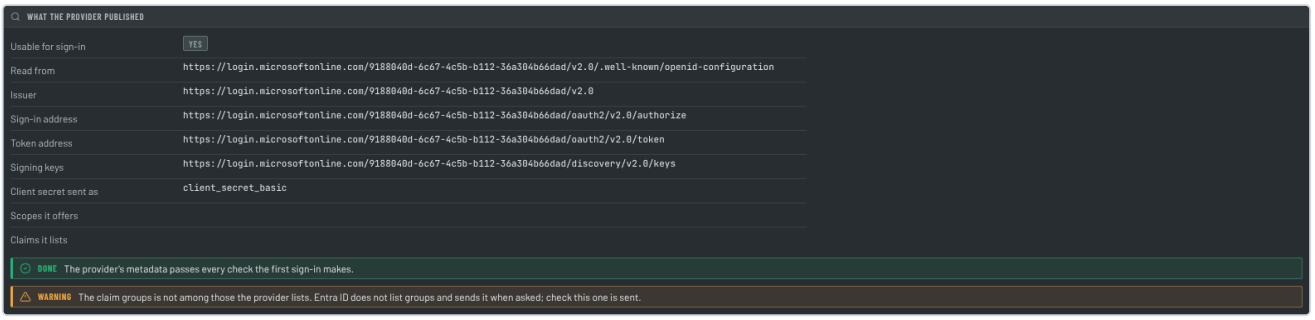
Column	Meaning
Provider group (required)	The group's name as the groups claim carries it
Service account (required)	The service account whose permissions, signal scope and limits govern what the group's members read of the plant
Roles	Comma-separated, for example `Operator, Engineer`
Screen permissions	Comma-separated screen permission names
Area	The area their page assignments are made for

A person in several groups gets every role and permission of all of them, and the service account of the first row that names one of their groups. Somebody no group covers is granted nothing. With no row at all, the table says *No group grants anything yet: nobody could sign in through the provider.*. While **Offer sign-in through the provider** is ticked, saving is refused without at least one row ("Add at least one group mapping: without one, nobody signing in through the provider is granted anything."), without an HTTPS **Authority (issuer)** or a **Client id**, and with a row missing its group or its service account. A block that is switched off can be saved half-written.

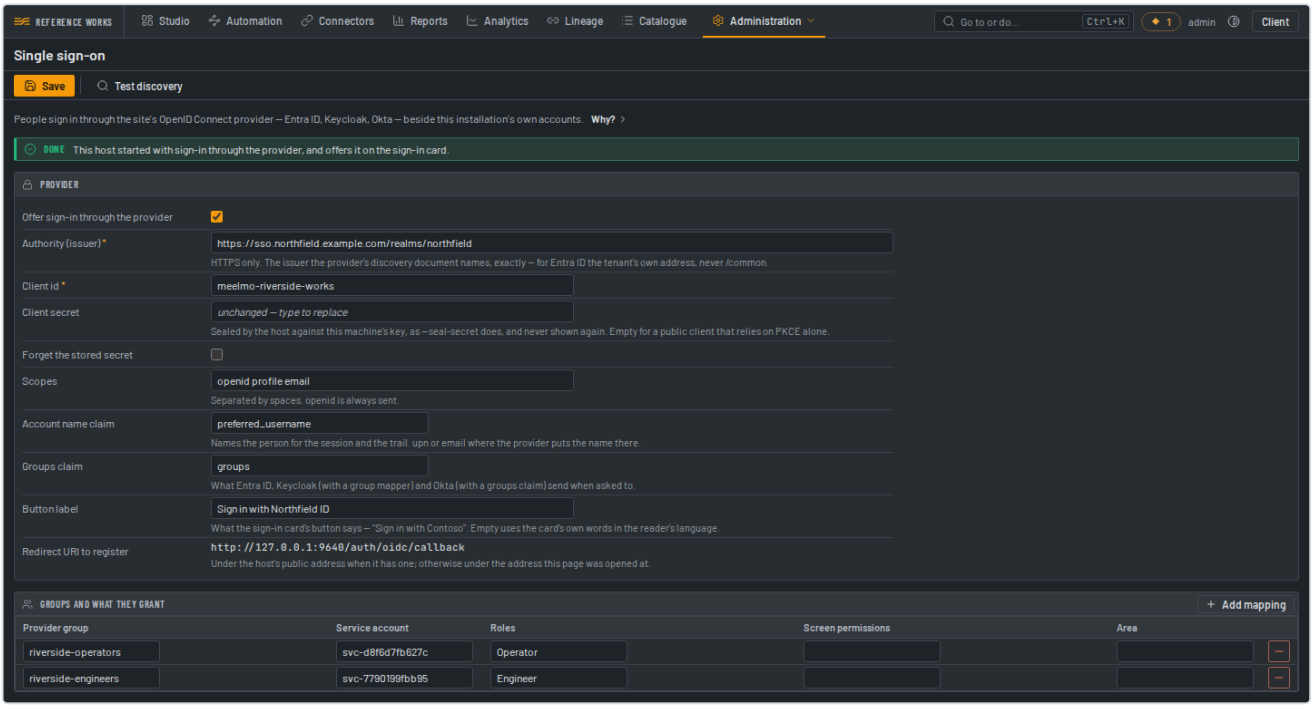
Test discovery reads the provider's metadata for the authority in the box, saved or not, and checks it as the first sign-in would. Nothing is saved. The answer appears in the panel **What the provider published**:

Field	Shows
Usable for sign-in	**Yes** or **No**. With **No**, a banner gives the reason (for example, the issuer is not the authority, or an endpoint is not HTTPS)
Read from	The discovery address that was read
Issuer, **Sign-in address**, **Token address**, **Signing keys**	What the provider published, when usable
Client secret sent as	`client_secret_post` or `client_secret_basic`
Scopes it offers, **Claims it lists**	The provider's own lists, where it publishes them

When usable, a banner says *The provider's metadata passes every check the first sign-in makes.* A scope or a claim you typed that the provider does not list gets a warning of its own. That is not necessarily wrong: a provider need not list every scope, and Entra ID does not list `groups` but sends it when asked.

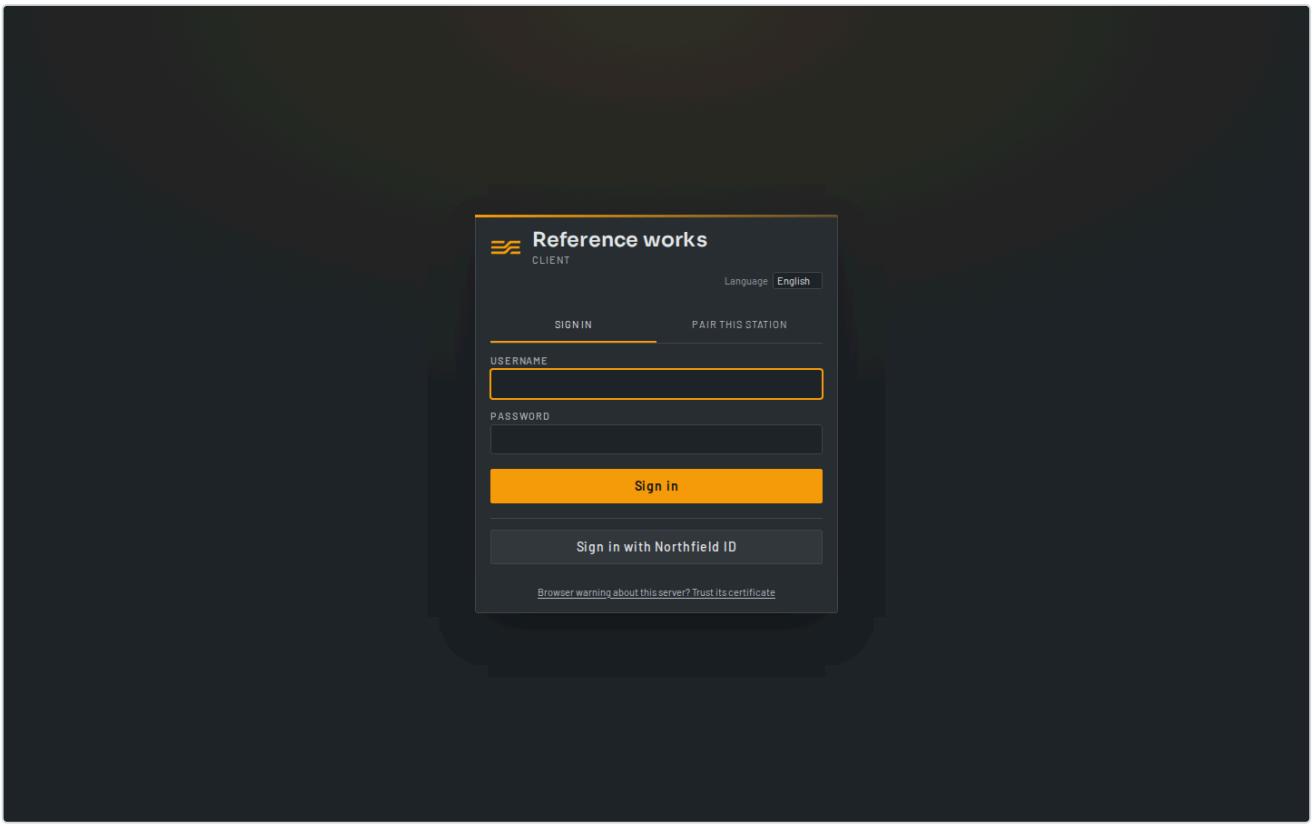


*What the provider published, for a Microsoft Entra ID authority typed in and tested: Usable for sign-in Yes, the discovery address read, the issuer, sign-in, token and signing-key addresses, client_secret_basic, the banner saying the metadata passes every check, and a warning that the groups claim is not among those the provider lists. In this release the *Scopes it offers* and *Claims it lists* rows draw empty although the provider publishes both.*



Single sign-on on the reference works: the host started with sign-in through the provider; authority https://sso.northfield.example.com/realms/northfield, client id meelmo-riverside-works with a stored secret, scopes openid profile email, the preferred_username and groups claims, the button label Sign in with Northfield ID and the redirect URI; the provider groups riverside-operators and riverside-engineers mapped to two service accounts with the roles Operator and Engineer

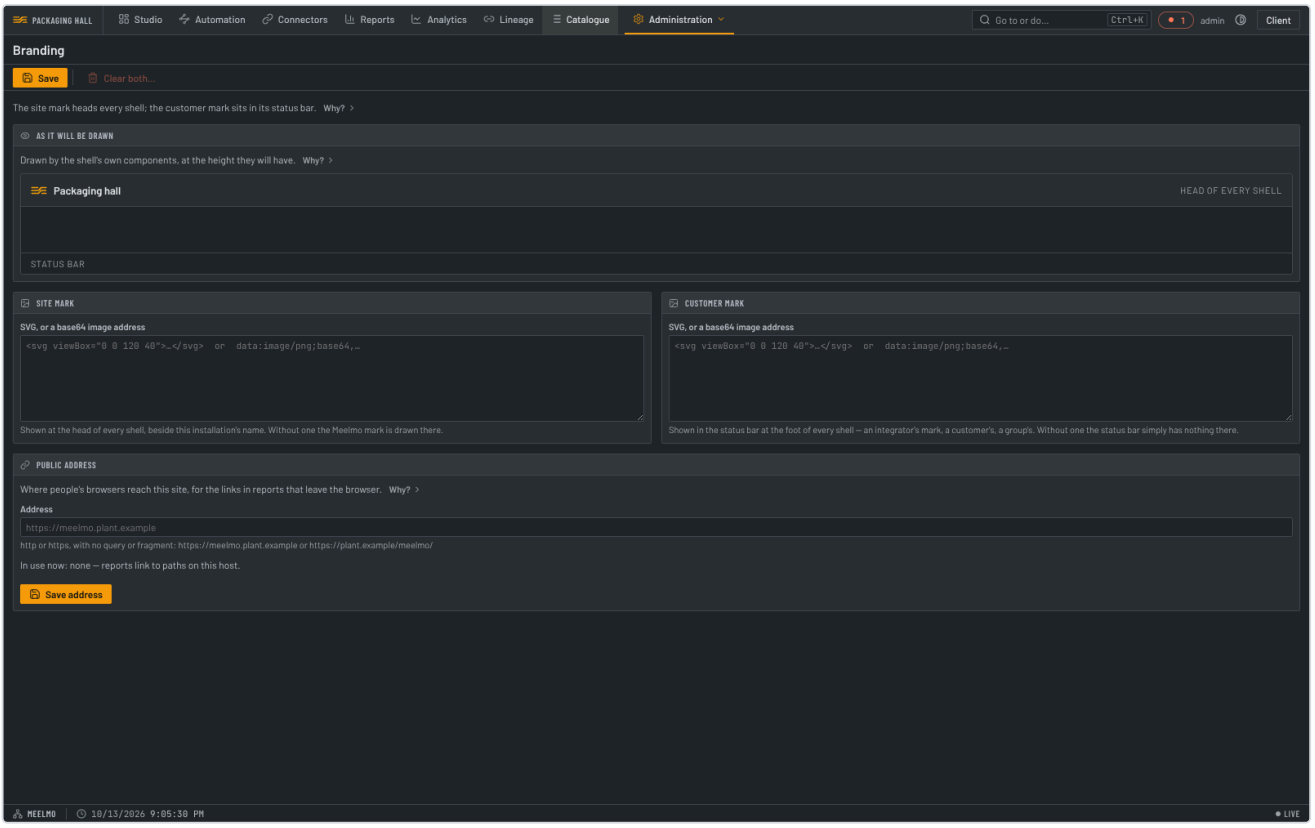
The sign-in card then offers the provider's button below the password form.



The sign-in card of the reference works with the Username and Password fields, Sign in, and below them Sign in with Northfield ID

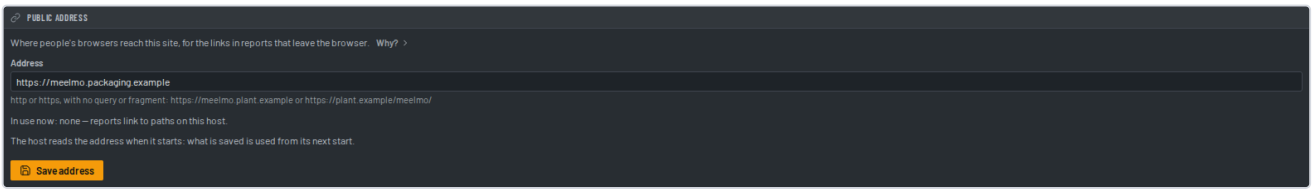
Branding

Administration ▸ Branding puts two marks on this installation's own screens: "The site mark heads every shell; the customer mark sits in its status bar."



Branding: the preview of the shell head and status bar, and the two boxes for the site mark and the customer mark, both empty.

Part	Meaning
As it will be drawn	the real shell components at the height the marks will have: **Head of every shell** and **Status bar**
Site mark — SVG, or a base64 image address	shown at the head of every shell beside the installation's name; without one the Meelmo mark is drawn
Customer mark — SVG, or a base64 image address	shown in the status bar at the foot of every shell — an integrator's, a customer's or a group's mark; without one nothing is drawn there
Save	stores both; "Every shell in this browser is already drawing it."
Clear both…	asks first, then removes both marks
Public address — Address	the address people's browsers reach this site at, which the links in reports that leave the browser are written against; with its own **Save address**. See *Configuration and deployment ▸ The public address*



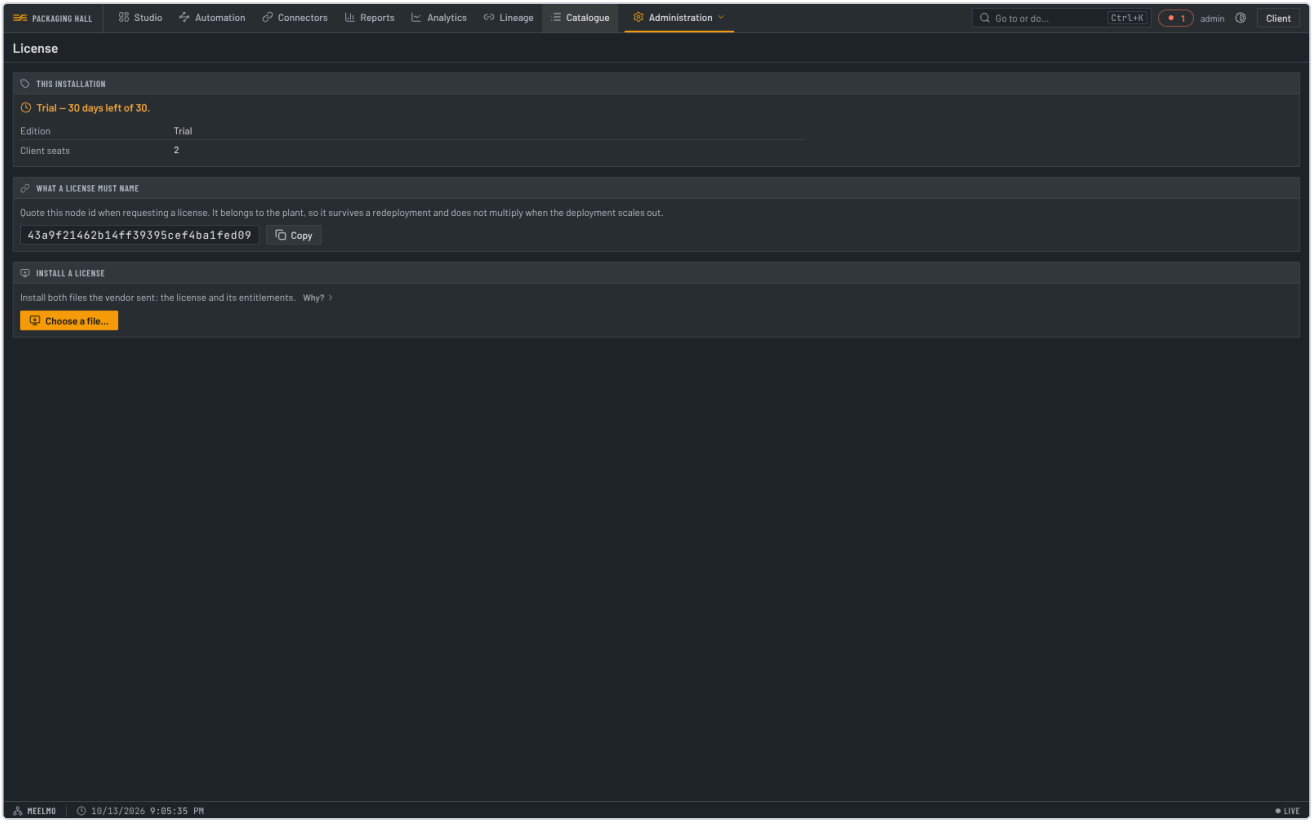
The Public address panel on Branding with https://meelmo.packaging.example typed in and saved: what it is for, the address box with its hint, In use now saying the host started with none, and that the saved address is used from the host's next start.

A mark is either an SVG drawing (`<svg viewBox="0 0 120 40">…</svg>`) or a base64 image address (`…`). Anything that fetches, scripts or will not parse is discarded ("None of this could be used…").

Example. Paste the plant's logo as an SVG with `viewBox="0 0 120 40"` into **Site mark**, check the preview, **Save**. The operator screens on the line show it at their next page load.

License

Administration ▸ License shows what the installation is licensed for and installs a licence.



License on a trial installation: thirty days left of thirty, edition Trial, two client seats, the node id to quote with a Copy button, and Choose a file to install a licence.

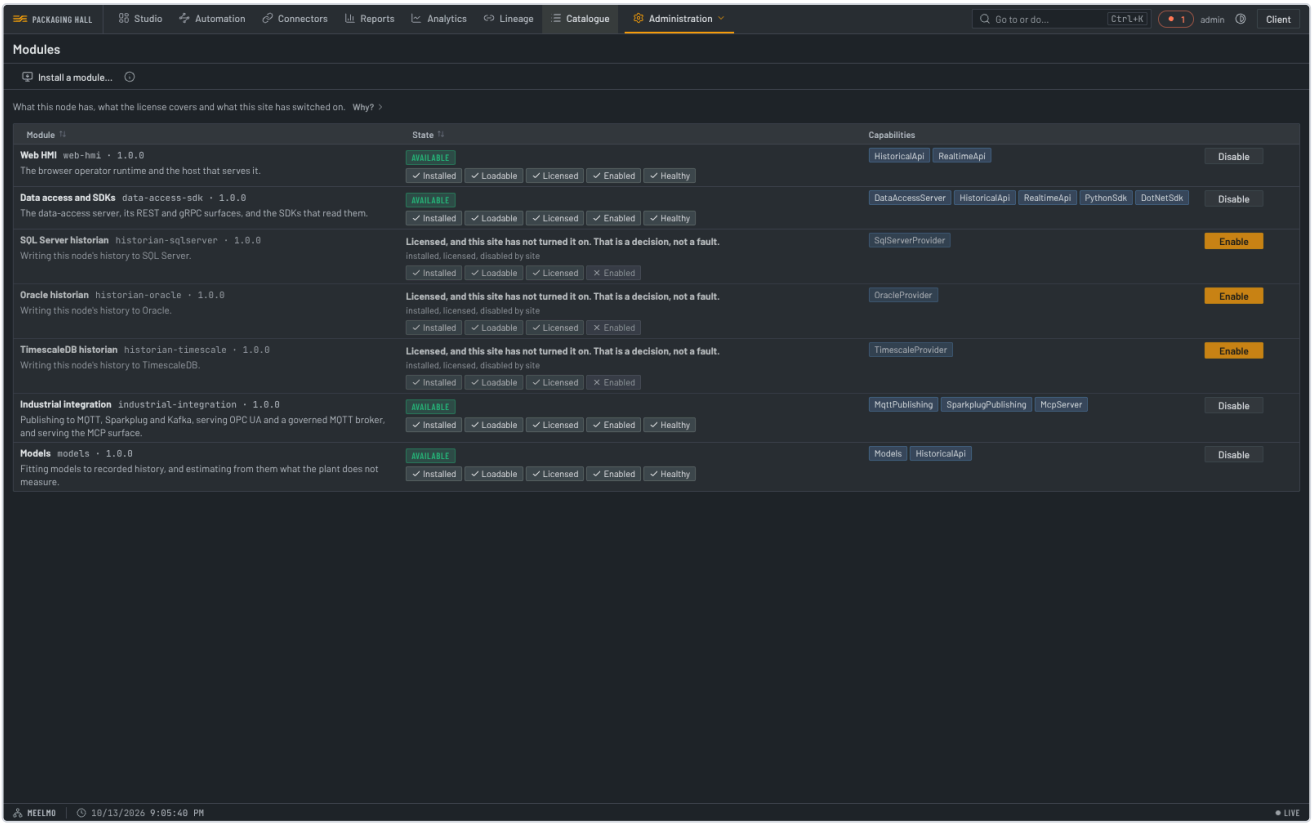
Panel	Shows
This installation	the state in one line — for example "Trial — 30 days left of 30.", "Licensed to … (…) — expires …", "Community Edition — free, no license required.", or why a licence does not apply — then **Edition**, **Licensed to**, **Expires** (or **Perpetual**) and **Client seats**
What a license must name	the id to quote when requesting a licence, with **Copy**. On a node this is the **node id**: it belongs to the plant, survives a redeployment and does not multiply when the deployment scales out. Elsewhere it is the machine id
Install a license	**Choose a file…** — pick the `.meelmolic` licence or the `.opsent` entitlements file. Install both: a licence without its entitlements grants no module

Modules and the browser HMI follow a new licence at once. The data-access server and publication were composed when the host started, so restart the host for those.

Example. Click **Copy** beside the node id and send it with the licence request. When the vendor's two files arrive, **Choose a file…** for the `.meelmolic`, then again for the `.opsent`, and check that **This installation** names the licensee.

Modules

Administration ▸ Modules shows "what this node has, what the license covers and what this site has switched on".



Modules: Web HMI, Data access and SDKs, Industrial integration and Models available; the three database historians licensed but not switched on, each with Enable.

Column	Shows
Module	name, identifier and version (for example **Web HMI** `web-hmi · 1.0.0`) and what it is for
State	**AVAILABLE**, or a sentence saying what is missing — "Licensed, and this site has not turned it on. That is a decision, not a fault." — then five checks: **Installed**, **Loadable**, **Licensed**, **Enabled**, **Healthy**
Capabilities	what the module provides, for example `HistoricalApi`, `RealtimeApi`, `DataAccessServer`, `PythonSdk`, `DotNetSdk`, `SqlServerProvider`, `MqttPublishing`, `SparkplugPublishing`, `McpServer`, `Models`
action	**Enable** or **Disable**

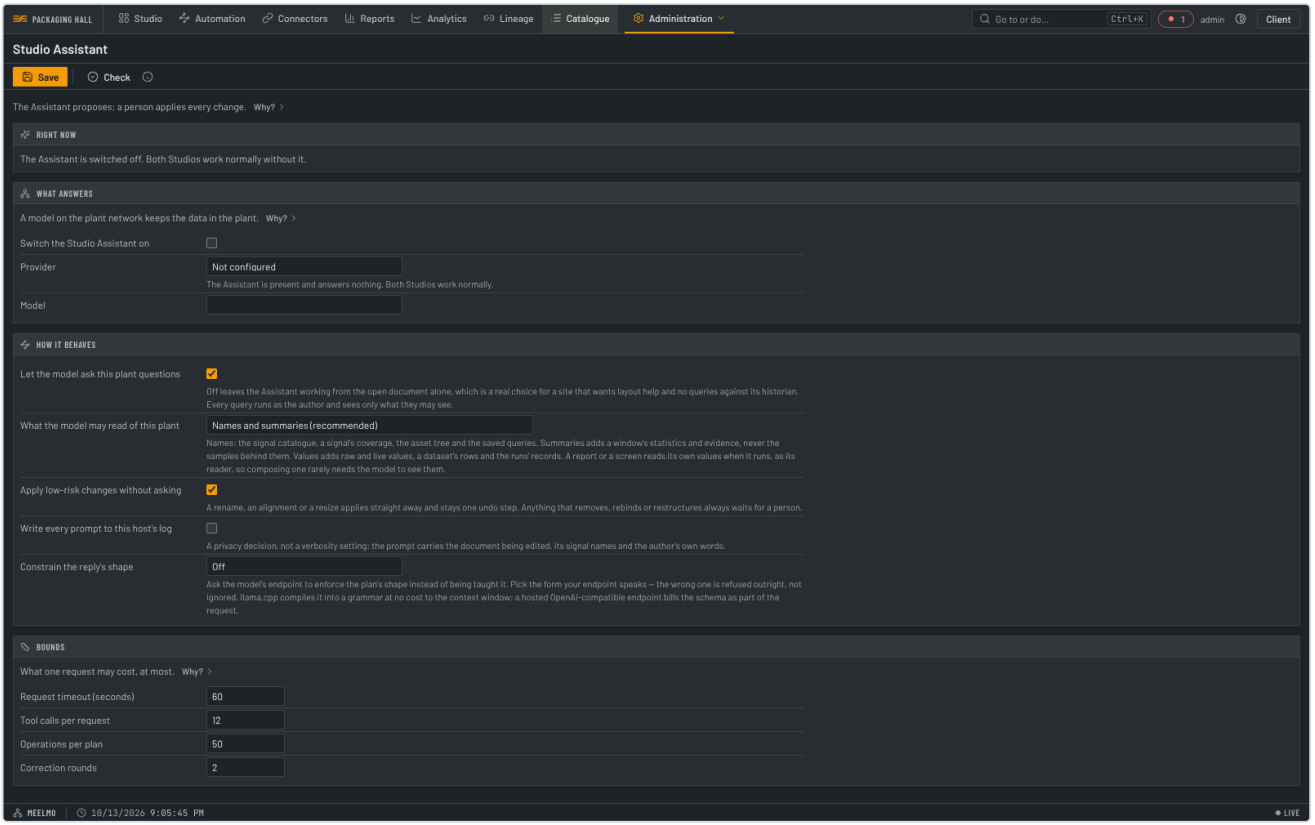
Other state sentences: not installed on this node; installed but this build cannot load it; not included in the licence (it stays visible so you can see what it would need); enabled and not working (the detail says what the node reported).

Enable and **Disable** activate a configuration revision, so each is recorded and can be rolled back. **Install a module…** takes a signed `module.json` for a module you have bought; it is installed beside the node, switched off — turning it on stays a separate decision.

Example. To store history in SQL Server: on **SQL Server historian** click **Enable**, then open **Historian**, choose **Microsoft SQL Server** as the store, fill in its connection, **Test**, **Validate**, **Create schema…** and **Save**.

Studio Assistant

Administration ▸ Studio Assistant decides what answers when an author asks the HMI Studio or the Report Studio to build or change something. "The Assistant proposes; a person applies every change."



Studio Assistant switched off: Right now says both Studios work normally without it; What answers with the switch, Provider Not configured and Model; How it behaves; and Bounds with the default limits.

The toolbar has **Save** and **Check**. **Check** asks the model one throwaway question — no document, no catalogue, no tools — and says what came back ("Answered by …"). Saving does not connect.

Section	Field	Meaning	Default
Right now	—	whether the Assistant is answering, and with what; "The Assistant is switched off. Both Studios work normally without it."	
What answers	**Switch the Studio Assistant on**	the switch	off
	Provider	**Not configured** (present, answers nothing), a hosted provider's own API, **OpenAI-compatible endpoint** (llama.cpp, vLLM, Ollama, LM Studio or a gateway of the site's own) or **Built-in test provider** (answers from a fixed script, to prove the panel, permissions and audit before a model exists)	Not configured
	Endpoint	the model's address including its path (OpenAI-compatible)	
	Model	the model name	
	API key	for a provider that needs one. Sealed against this installation when saved and never sent back to the screen ("stored — leave blank to keep it"); **Remove the stored credential** clears it	

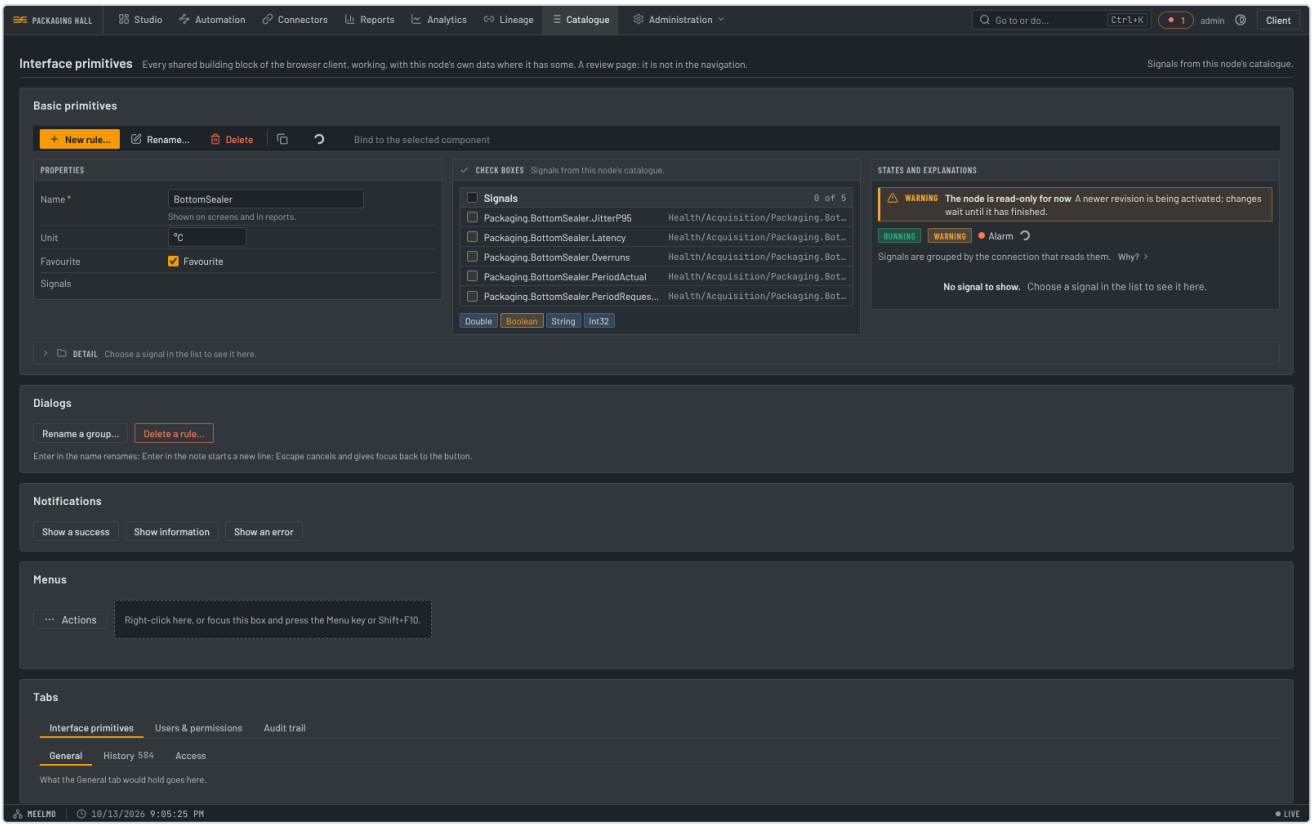
Section	Field	Meaning	Default
How it behaves	**Let the model ask this plant questions**	off leaves the Assistant working from the open document alone; every query runs as the author	on
	What the model may read of this plant	**The names of what exists**, **Names and summaries (recommended)**, or **Values too: samples, live values, dataset rows**	Names and summaries
	Apply low-risk changes without asking	a rename, an alignment or a resize applies at once as one undo step; anything that adds, removes, rebinds or restructures waits for a person	on
	Write every prompt to this host's log	a privacy decision: the prompt carries the document, its signal names and the author's words	off
	Constrain the reply's shape	ask the endpoint to enforce the plan's shape (**Off**, or the form the endpoint speaks)	Off
Bounds	**Request timeout (seconds)**		`60`
	Tool calls per request		`12`
	Operations per plan		`50`
	Correction rounds		`2`

A model on the plant network answers without any data leaving the plant; a hosted provider sends the document being edited outside it. Fields set by environment variables are listed and keep winning over what is saved here. Where the licence does not include the Studio Assistant, the page says so. The Studio Assistant module's own chapter, where licensed, describes the Assistant in use.

Example: a model on the plant network. Tick **Switch the Studio Assistant on**, **Provider** *OpenAI-compatible endpoint*, **Endpoint** `http://ai-gw.plant.local:8080/v1/chat/completions`, **Model** as the server names it, leave **What the model may read of this plant** at *Names and summaries (recommended)*. **Check**; when it answers, **Save**. "Saved. The next question goes to this model; no restart is needed."

Interface primitives

`/admin/interface` — **Interface primitives** — shows every shared building block of the browser client working with this node's own data where it has some: buttons and fields, check boxes over the node's signal catalogue, states and explanations, dialogs, notifications, menus and tabs. It is a review page for administrators and integrators judging how the client behaves (including by keyboard); it is not in the navigation and changes nothing on the node.



The Interface primitives review page: a property panel, a check-box list of this node's signals, states and explanations, dialog, notification and menu examples, and nested tabs.

First-run setup (`/setup`)

An installation with nothing configured — no node, no account, no user — opens `/setup` instead of a sign-in page. It collects everything in one form and applies it in one call, because a half-configured installation is worse than an unconfigured one. It is in the browser's language.

Section	Field	Meaning
This installation	**Site name**	the name shown on every screen, for example `Plant 1`
Administrator	**Username**, **Password** (required)	on a node with no administrator, this creates one — on the node, where the Desktop and the browser both read it. On a node that already has accounts, give one of them and its own password
Data	**Node database**	leave empty for the node beside this installation's state; **Find nodes on this network** lists nodes that announce themselves
	Node name (required for a new node)	naming a never-commissioned node brings it into service, for example `Line 3 Recorder`
	Node claim (required for a remote, never-commissioned node)	the one-time claim that node's own host printed when it started
Network	**Bind address**	default `127.0.0.1`
	Port	default `8443`
	TLS certificate (PFX)	required for any address other than `127.0.0.1`; without it a browser does not return the session cookie and signing in appears to do nothing

Section	Field	Meaning
	This host is behind a TLS-terminating proxy	when TLS ends at a proxy in front of the host
	Serve this node to other workstations, **Live data port**	lets Desktop workstations administer the node and read its live samples on that port

Configure this installation applies it; the line beside it says what is still needed ("Still needed: a username, a password."). The host then restarts to pick up its node, network and certificate, and the page says **Configured** — **Ready.** with **Go to the client**, or asks you to restart the service first. A host that must be configured from its own machine asks for the setup key printed on its console.

Once anybody can sign in, `/setup` closes and says **Already set up**; accounts are then added under **Administration ▸ Users & permissions**.

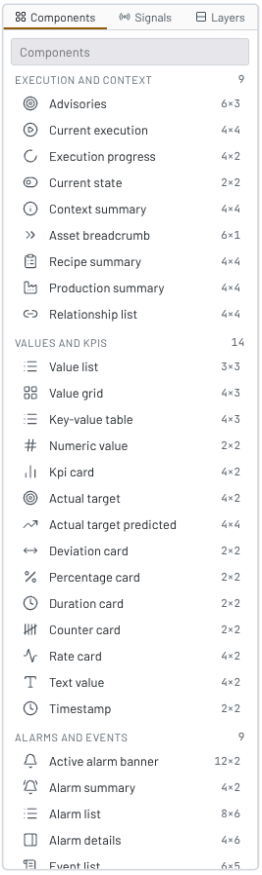
HMI component reference

> **Module: Web HMI.** This chapter describes the components of the HMI Studio, supplied by the Web HMI module. It is only present on installations licensed for that module.

This chapter is the reference for every component you can place on an HMI page: what it shows, where it can go, what you bind to it and every setting its inspector offers. Use it beside *The HMI Studio* chapter (*21-web-hmi-studio.md*), which explains how to place, bind and publish, and *The browser operator HMI* (*20-web-operator-hmi.md*), which shows the components in use.

The catalogue holds 103 component types in 10 categories, and the Symbol shape draws from a library of 117 equipment symbols. Of the palette's types outside **Shapes and symbols**, 60 of the 93 types are grid-only and 9 of the process-mimic types are canvas-only.

The entries are written from the code. Each one below is produced from the same catalogue the Studio's palette, the inspector, the server's validator and the operator screen read, so the names, slots, defaults and ranges are the ones the Studio offers. The prose between the entries (how to choose, how a few components behave, the limits of this release and the worked examples) is written by hand.



The Components palette, by category, each type with its default size in grid cells

How to read an entry

Each entry has the name the palette shows, with the type id in brackets. The id is what a page document stores, what the inspector shows at its top, and what you can type in the **Components** box to find the type. Under the name comes the sentence the palette shows when you rest the pointer on the type, and then:

Line	What it tells you
Places on	**Dashboard grid** with the size the component takes when you drop it and the smallest it may be made, in cells of the page's grid; **Process canvas** with the size it is drawn at when you drop it, in canvas units. A type that lists only one cannot be placed on the other: the palette does not offer it there.
Updates	Its update class: **On change**, **Fast**, **Moderate**, **Detailed** or **Not live**. The classes and their rates are described in *The HMI Studio ▸ Update classes*.
History fetched when the page opens	Only on components that draw history. How many points of history the page asks for on this component's behalf when it opens. The **Page cost** tab adds these up for the whole page.
Data	Its binding slots, as the inspector's **Data** section lists them: the slot's name, the kinds of data it takes (**Numeric**, **Boolean**, **Text**, **Entity**, **Plottable** for numeric or boolean, **Any** for all of them), *several, in order* where the slot takes a list, and *required* where the page does not validate until it is bound.
The settings table	Every setting the type declares: its label in the inspector, the inspector section it is in, its value when you drop the component, and what it accepts. A dash means the setting starts empty.

The settings tables list only what is particular to a type. Every component also has the inspector's common sections, described in *The HMI Studio ▸ The inspector*: **General** (title, subtitle, description, **Critical — cannot be hidden by a rule**), **Thresholds**, the common **Formatting** and **Appearance** fields, **Navigation**, **Visibility**, **Responsive** and **Advanced**. A drawn shape has no binding slots; its data arrives through the animations listed under **Driven by**.

Paints. Shapes, symbols and a machine model's parts are painted from one set of named paints rather than from free colours, so a drawing follows the theme and a state colour always means the same thing: None, Surface, Outline, Text, Information, Running, Ready, Completed, Attention, Warning, Alarm, Critical, Unavailable, Disabled, Stale, Bad Quality, Disconnected and Selected. Where an entry says *a paint*, the setting offers this list.

Choosing a component

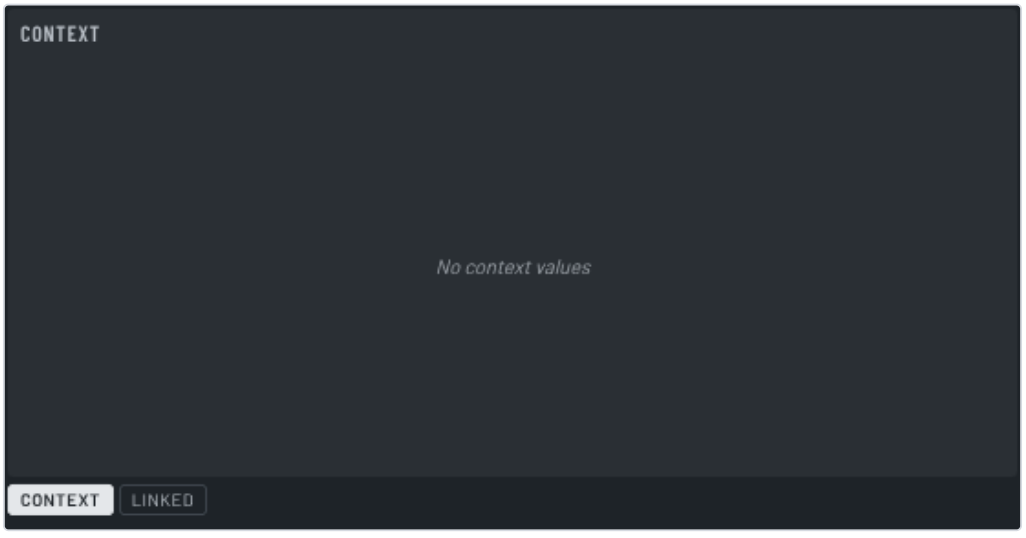
Start from the question the operator asks, then pick the smallest component that answers it.

The operator asks	Use	Rather than
What is this one value now?	**Numeric value**, or **Process sensor** on a canvas	A gauge, unless the position in a range matters
How do eight readings of one machine look now?	**Value list** or **Value grid**	Eight numeric values: eight borders and headers for the same facts
Am I on target?	**Actual target**, **Deviation card**, **Kpi card**	Two numeric values side by side
Where in its range is it?	**Linear gauge**; **Vertical gauge** for a level	**Radial progress**, which is for bounded progress and utilisation
Is it inside its acceptable band?	**Range gauge**	A gauge with thresholds that hides the band
What state is it in, and for how long?	**State indicator** (a signal) or **Current state** (the semantic model's state)	A traffic light, which has three states only
What did it do recently?	**Mini trend** beside a labelled value; **Live trend** for a real chart	A historical trend, which is not live

The operator asks	Use	Rather than
What did it do over the shift?	**Historical trend**, **Statistics card**, **Distribution chart**	A live trend with a long window
What is the waveform of a fast signal?	**Live scope**	A live trend, whose time axis moves too slowly to show the shape
Did several signals move together?	**Stacked trend** when they share a clock; **Multiplot** to compare many side by side	A multi-signal trend once its traces cross
Where did the line's time go?	**Line account**	Blocked and starved boolean tracks read by eye
Is this run like the good ones?	**Run overlay**	A historical trend of one run
What is wrong now?	**Active alarm banner** (mark it critical), **Alarm list**	An event list, which records what happened
What should be done about the line?	**Advisories**	An alarm, which is about an instant
How do I get to the detail?	**Page link**, **Asset link**, **Navigation hotspot**, or a component's own **Navigation**	Text that says where to go

Containers

Containers arrange a page; none of them reads data. On a dashboard grid each component is its own cell of the page: a panel or a card placed under other components is a surface drawn beneath them, not a box that owns them. That is why **Tabs** works through a page parameter (below) rather than by holding its views.



Two containers on the reference works' Batch page: the Context summary that the Tabs component shows, here with No context values, and the Tabs row under it with Context selected and Linked beside it

Section (`section`)

A titled band that groups the components under it.

- **Places on:** Dashboard grid, 12×1 cells (at least 2×1)
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Heading	General	—	text
Show a dividing rule	Appearance	on	on or off

Panel (`panel`)

A bordered surface. Use it to separate one topic from another.

- **Places on:** Dashboard grid, 4×4 cells (at least 2×2); Process canvas, 160 × 100
- **Updates:** Not live
- **Data:** no binding slots

No settings of its own.

Card (`card`)

A titled card with a body.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1); Process canvas, 160 × 100
- **Updates:** Not live
- **Data:** no binding slots

No settings of its own.

Tabs (`tabs`)

A row of tabs that sets a page parameter. Give the components of each view a visibility rule reading it, or name the parameter in their signal ids. Write a tab as value|Label to differ the two.

- **Places on:** Dashboard grid, 12×1 cells (at least 2×1)
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Tabs	General	—	a list, comma separated
Page parameter set	Data	`view`	text

Accordion (`accordion`)

Collapsible sections for detail an operator opens on demand.

- **Places on:** Dashboard grid, 6×4 cells (at least 3×2)
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Sections	General	—	a list, comma separated
Page parameter set	Data	`view`	text

Spacer (`spacer`)

Deliberate empty space, so a layout's gaps are intentional.

- **Places on:** Dashboard grid, 2×1 cells (at least 1×1)
- **Updates:** Not live
- **Data:** no binding slots

No settings of its own.

Divider (`divider`)

A rule between groups.

- **Places on:** Dashboard grid, 12×1 cells (at least 1×1); Process canvas, 160 × 100
- **Updates:** Not live
- **Data:** no binding slots

No settings of its own.

How tabs switch a view

Tabs draws a row of tab buttons. Pressing one sets a page parameter (**Page parameter set**, `view` unless you change it) to that tab's value, and the tab whose value the parameter holds is drawn as selected. The tab does not hide anything itself: give each component that belongs to a view a **Visibility** rule of `ParameterEquals` reading that parameter. Write each entry of **Tabs** as `value|Label` when the stored value and the word on the tab should differ, for example `oee|Efficiency`; an entry without `|` is both.

With no tabs listed, the component says *No tabs configured*. Page parameters are described in *HMI assets, expressions and parameters*.

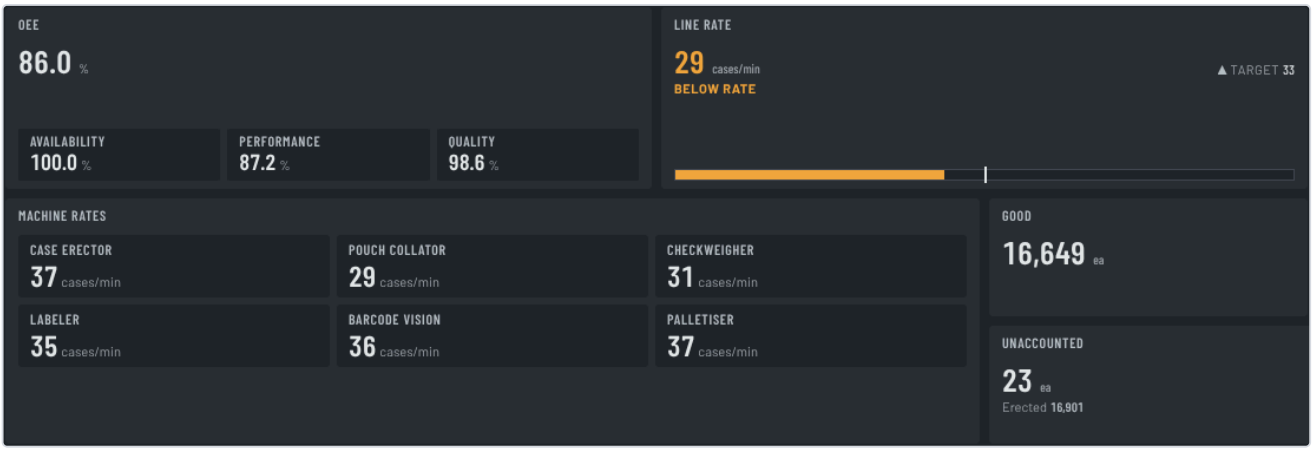
Accordion works the same way: each entry of **Sections** is a button that sets **Page parameter set**, and the components of a section carry the visibility rule. With no sections listed, it says *No tabs configured*.

Settings the value components share

Show timestamp and **Show quality** are in **Appearance** for every component, not among a component's own settings. **Show unit** and **Compact large numbers** apply to every value card, gauge and process sensor that offers them: with **Show unit** off the reading is written without its unit, and **Compact large numbers** writes 12 400 as `12.4k`. A **Progress indicator** with **Show percentage** on writes how far its value is toward **Maximum** (`50 %`) rather than the value itself. An **Icon**'s **Colour** is the colour it is drawn in; without one it takes **Appearance ▸ Background**, and then the information colour. An **Execution timeline** with **Show status** off draws every run in one colour and leaves the status out of its tooltip.

Values and KPIs

These show numbers and words now. The cards share their slot names (**Actual value**, **Target value**), so you can swap one for another and keep the bindings you made.



Values on the packaging line's Overview page: the OEE KPI card with availability, performance and quality as tiles under it, the Line rate actual target below its target with the BELOW RATE band, the Machine rates value grid of six machines' throughput as stat tiles inside one panel, and the Good and Unaccounted numeric values, the second with its caption line

Value list (`value-list`)

A stack of label-and-value rows: many readings in the space of one card.

- **Places on:** Dashboard grid, 3×3 cells (at least 2×1)
- **Updates:** Fast
- **Data:**
  - **Signals**: Any, several, in order

Setting	Section	Default	Accepts
Decimal places	Formatting	1	a whole number from 0 to 9
Show unit	Appearance	on	on or off
Mark values that are not live	Appearance	on	on or off
Short names	Appearance	on	on or off

Value grid (`value-grid`)

Readings in columns, each with its label above it. The densest way to show a set.

- **Places on:** Dashboard grid, 4×3 cells (at least 2×1)
- **Updates:** Fast
- **Data:**
  - **Signals**: Any, several, in order

Setting	Section	Default	Accepts
Columns	Appearance	2	a whole number from 1 to 4
Decimal places	Formatting	1	a whole number from 0 to 9
Show unit	Appearance	on	on or off
Mark values that are not live	Appearance	on	on or off
Short names	Appearance	on	on or off

Key-value table (`key-value-table`)

Label, right-aligned figure and unit, a row each — or a grid of readings under column headings.

- **Places on:** Dashboard grid, 4×3 cells (at least 2×2)
- **Updates:** Fast
- **Data:**
  - **Readings, in table order**: Any, several, in order

Setting	Section	Default	Accepts
Decimal places	Formatting	1	a whole number from 0 to 9
Show the unit column	Appearance	on	on or off
Column headings	Data	—	a list, comma separated. Leave empty for one reading a row. With headings, the readings fill the table row by row, as many to a row as there are headings.
Row labels	Data	—	a list, comma separated. A label for each row. Without them a row takes its first reading's name; a label that is a key in the project's string table is translated.

Numeric value (`numeric-value`)

One number, large, with its unit and quality, and an optional caption line.

- **Places on:** Dashboard grid, 2×2 cells (at least 2×1); Process canvas, 150 × 80
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required
  - **Caption value**: Any

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off
Show a mini trend	Appearance	off	on or off
Caption under the reading	General	—	text

Kpi card (`kpi-card`)

A headline number with its target and direction of travel, and the readings it is made of as tiles under it.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×2)
- **Updates:** Fast
- **History fetched when the page opens:** up to 120 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Actual value**: Numeric, required
  - **Target value**: Numeric

- **Trend signal**: Plottable
- **Readings it is made of, as tiles under it**: Any, several, in order

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off
Show a mini trend	Appearance	off	on or off
Show change against target	Appearance	on	on or off

Actual target (`actual-target`)

Actual against target, with the gap named rather than implied.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×2)
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required
  - **Target value**: Numeric

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off
Show a mini trend	Appearance	off	on or off
Show the comparison bar	Appearance	on	on or off

Actual target predicted (`actual-target-predicted`)

Actual, target and prediction, drawn so the three stay distinguishable without colour alone.

- **Places on:** Dashboard grid, 4×4 cells (at least 3×2)
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required
  - **Target value**: Numeric
  - **Predicted value**: Numeric

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off

Setting	Section	Default	Accepts
Show a mini trend	Appearance	off	on or off
Show the comparison bar	Appearance	on	on or off

Deviation card (`deviation-card`)

The signed gap between a value and its target.

- **Places on:** Dashboard grid, 2×2 cells (at least 2×2)
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required
  - **Target value**: Numeric

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off
Show as a percentage	Appearance	off	on or off

Percentage card (`percentage-card`)

A ratio as a percentage, with the two operands available beneath.

- **Places on:** Dashboard grid, 2×2 cells (at least 2×2)
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required
  - **Total**: Numeric

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off

Duration card (`duration-card`)

A duration, written the way a person would say it.

- **Places on:** Dashboard grid, 2×2 cells (at least 2×2)
- **Updates:** On change
- **Data:**
  - **Duration in seconds**: Numeric, required

No settings of its own.

Counter card (`counter-card`)

A running count, grouped so six digits stay readable.

- **Places on:** Dashboard grid, 2×2 cells (at least 2×2)
- **Updates:** On change
- **Data:**
  - **Actual value**: Numeric, required
  - **Target value**: Numeric

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off
Show a mini trend	Appearance	off	on or off
Group thousands	Formatting	on	on or off

Rate card (`rate-card`)

A rate with its unit of time, and the recent shape of it.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×2)
- **Updates:** Fast
- **History fetched when the page opens:** up to 120 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Actual value**: Numeric, required
  - **Target value**: Numeric
  - **Trend signal**: Plottable

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off
Show a mini trend	Appearance	off	on or off
Per unit	Formatting	`h`	text

Text value (`text-value`)

A string signal or context value, shown as text.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1); Process canvas, 160 × 100
- **Updates:** On change
- **Data:**
  - **Text value**: Any, required

No settings of its own.

Timestamp (`timestamp`)

When something last happened, absolute or relative: the last reading, or the last time the value changed — what a watchdog tile needs, since a polled source stamps every reading. Counting from the last change, its thresholds compare the age in seconds.

- **Places on:** Dashboard grid, 2×2 cells (at least 2×1)
- **Updates:** On change
- **Data:**
  - **Timestamp source**: Any

Setting	Section	Default	Accepts
Show as time since	Formatting	on	on or off
Time of	Data	The last reading	one of: The last reading, The last change of value

Gauges and indicators



Vertical gauges on the packaging line's Command page: rejects by cause for the shift, and the case weight, seal temperature, label stock and air pressure, each with its reading under the bar

Linear gauge (`linear-gauge`)

A horizontal bar with the value written out. Preferred over a dial for an ordinary analogue value.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1); Process canvas, 220 × 60
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required
  - **Target value**: Numeric
  - **Minimum**: Numeric
  - **Maximum**: Numeric

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off
Scale minimum	Data	0	a number
Scale maximum	Data	100	a number
Show the scale	Appearance	on	on or off

Setting	Section	Default	Accepts
Show the reading	Appearance	on	on or off

Vertical gauge (`vertical-gauge`)

A vertical bar. Right for levels, tanks and anything with a real up.

- **Places on:** Dashboard grid, 2×4 cells (at least 1×3); Process canvas, 70 × 200
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required
  - **Target value**: Numeric
  - **Minimum**: Numeric
  - **Maximum**: Numeric

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off
Scale minimum	Data	0	a number
Scale maximum	Data	100	a number
Show the reading	Appearance	on	on or off

Radial progress (`radial-progress`)

A ring for bounded progress and utilization. Not for every analogue value — geometry alone reads badly at a distance.

- **Places on:** Dashboard grid, 2×2 cells (at least 2×2); Process canvas, 140 × 140
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required
  - **Target value**: Numeric
  - **Maximum**: Numeric

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off
Show percentage	Appearance	on	on or off

Range gauge (`range-gauge`)

A value against an acceptable band, with the band drawn.

- **Places on:** Dashboard grid, 4×2 cells (at least 3×2)
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required
  - **Low limit**: Numeric
  - **High limit**: Numeric
  - **Minimum**: Numeric
  - **Maximum**: Numeric

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off

Segmented gauge (`segmented-gauge`)

Discrete segments, readable from further away than a smooth bar.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1); Process canvas, 160 × 100
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required

Setting	Section	Default	Accepts
Segments	Appearance	10	a whole number from 2 to 40
Scale minimum	Data	0	a number
Scale maximum	Data	100	a number

Traffic light (`traffic-light`)

Three states with their labels. Never colour alone.

- **Places on:** Dashboard grid, 1×3 cells (at least 1×2); Process canvas, 56 × 150
- **Updates:** On change
- **Data:**
  - **Value**: Plottable, required

No settings of its own.

State indicator (`state-indicator`)

The current state, named, with how long it has held.

- **Places on:** Dashboard grid, 2×2 cells (at least 1×1); Process canvas, 150 × 60
- **Updates:** On change
- **Data:**
  - **State source**: Any, required

Setting	Section	Default	Accepts
Show time in state	Data	on	on or off

Quality indicator (`quality-indicator`)

Data quality across a set of signals, so a bad sensor is visible.

- **Places on:** Dashboard grid, 2×2 cells (at least 1×1)
- **Updates:** On change
- **Data:**
  - **Signals to watch**: Any, several, in order

No settings of its own.

Progress indicator (`progress-indicator`)

Progress toward a known end.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1); Process canvas, 160 × 100
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required
  - **Maximum**: Numeric

Setting	Section	Default	Accepts
Show percentage	Appearance	on	on or off

Deviation from aim (`deviation-bar`)

Each reading's signed distance from its aim as a bar either side of zero, with OK, HIGH or LOW from the tolerance.

- **Places on:** Dashboard grid, 4×4 cells (at least 3×2)
- **Updates:** Fast
- **Data:**
  - **Readings**: Numeric, several, in order
  - **Aims, one for each reading in the same order**: Numeric, several, in order

Setting	Section	Default	Accepts
In range within ± this of the aim	General	0	a number, at least 0
Bar full scale (±); 0 fits the readings	Appearance	0	a number, at least 0
Deviation as a percentage of the aim	Formatting	off	on or off
Decimal places	Formatting	2	a whole number from 0 to 9
Show unit	Appearance	on	on or off

Step progress (`step-progress`)

A sequence's steps sized by their share, the current one marked, with a now line and the progress overall.

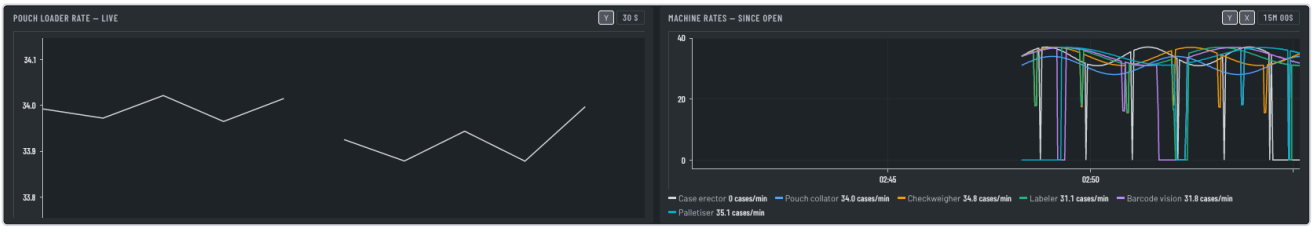
- **Places on:** Dashboard grid, 12×2 cells (at least 4×1)
- **Updates:** On change

- **Data:**
  - **Current step: its number, or its value**: Any, required
  - **Done so far**: Numeric
  - **Total**: Numeric

Setting	Section	Default	Accepts
Steps, in order	Data	—	a list, comma separated. One step a line, written value|Label|share. The value is what the current-step signal reports while the step is under way; the share sizes its segment.
Progress caption	General	—	text
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9

The gauges with **Minimum** and **Maximum** slots take their scale from those signals when they are bound, and from **Scale minimum** and **Scale maximum** when they are not. *The HMI Studio* chapter's worked example builds a **Linear gauge** for the case weight on the Overview page.

Trends and analytics



Charts on the packaging line's Trends page: a Live scope of the line's signals beside a Data logger of the machine rates since the page was opened

Mini trend (`mini-trend`)

A sparkline: shape without axes, for a value already labelled nearby.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1); Process canvas, 200 × 70
- **Updates:** Moderate
- **History fetched when the page opens:** up to 200 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Signal**: Plottable, required

Setting	Section	Default	Accepts
Window	Data	`15m`	a duration such as `15m`, `1h`, `24h`

Live trend (`live-trend`)

A live chart with a rolling window and explicit gaps.

- **Places on:** Dashboard grid, 6×5 cells (at least 3×3)
- **Updates:** Detailed

- **History fetched when the page opens:** up to 4,000 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Signals**: Plottable, several, in order
  - **Context: state**: Entity
  - **Context: run**: Entity

Setting	Section	Default	Accepts
Window	Data	`15m`	a duration such as `15m`, `1h`, `24h`
Show legend	Appearance	on	on or off
Show grid	Appearance	on	on or off
Show the time axis	Appearance	on	on or off
Point budget	Advanced	4,000	a whole number from 100 to 20,000
Retained samples	Advanced	500,000	a whole number from 10,000 to 2,000,000
Scroll continuously	Advanced	on	on or off
Run the axis past now by (minutes)	Appearance	0	a number from 0 to 1,440

Historical trend (`historical-trend`)

A historical chart, downsampled by the server with gaps preserved.

- **Places on:** Dashboard grid, 8×5 cells (at least 4×3)
- **Updates:** Not live
- **History fetched when the page opens:** up to 4,000 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Signals**: Plottable, several, in order
  - **Context: state**: Entity
  - **Context: run**: Entity

Setting	Section	Default	Accepts
Window	Data	`8h`	a duration such as `15m`, `1h`, `24h`
Show legend	Appearance	on	on or off
Show grid	Appearance	on	on or off
Show the time axis	Appearance	on	on or off
Point budget	Advanced	4,000	a whole number from 100 to 20,000
Retained samples	Advanced	500,000	a whole number from 10,000 to 2,000,000
Bucket value	Advanced	Default (time-weighted for on-change signals)	one of: Default (time-weighted for on-change signals), Time-weighted average, Sample mean, Minimum, Maximum

Multi signal trend (`multi-signal-trend`)

Several signals on one time axis. Units that differ get their own value axis unless told otherwise.

- **Places on:** Dashboard grid, 8×6 cells (at least 4×3)
- **Updates:** Detailed
- **History fetched when the page opens:** up to 4,000 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Signals**: Plottable, several, in order
  - **Context: state**: Entity
  - **Context: run**: Entity

Setting	Section	Default	Accepts
Window	Data	`1h`	a duration such as `15m`, `1h`, `24h`
Share one value axis	Appearance	off	on or off
Show legend	Appearance	on	on or off
Show grid	Appearance	on	on or off
Show the time axis	Appearance	on	on or off
Point budget	Advanced	4,000	a whole number from 100 to 20,000
Retained samples	Advanced	500,000	a whole number from 10,000 to 2,000,000
Bucket value	Advanced	Default (time-weighted for on-change signals)	one of: Default (time-weighted for on-change signals), Time-weighted average, Sample mean, Minimum, Maximum
Scroll continuously	Advanced	on	on or off
Run the axis past now by (minutes)	Appearance	0	a number from 0 to 1,440

Live scope (`live-scope`)

An oscilloscope for one fast signal: its waveform against sample position, swept or scrolled. For a vibration or a valve hunting, where a time axis moves too slowly to see the shape.

- **Places on:** Dashboard grid, 6×4 cells (at least 3×2)
- **Updates:** Detailed
- **History fetched when the page opens:** up to 4,000 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Signal**: Plottable, required

Setting	Section	Default	Accepts
Window	Data	`30s`	a duration such as `15m`, `1h`, `24h`
Sweep style	Appearance	Sweep	one of: Sweep, Scroll
Points shown (0 fits the window)	Appearance	0	a whole number from 0 to 20,000
Show the value axis	Appearance	on	on or off
Point budget	Advanced	8,000	a whole number from 100 to 20,000

Data logger (`data-logger`)

Signals accumulating from the moment the screen opened, the axis following the newest point. The whole ramp on one axis while a plant comes up — no window chosen in advance.

- **Places on:** Dashboard grid, 6×5 cells (at least 3×3)
- **Updates:** Detailed
- **History fetched when the page opens:** up to 4,000 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Signals**: Plottable, several, in order

Setting	Section	Default	Accepts
Window	Data	`15m`	a duration such as `15m` , `1h` , `24h`
Follow	Appearance	Grow to fit	one of: Grow to fit, Sliding window
Show legend	Appearance	on	on or off
Show grid	Appearance	on	on or off
Show the time axis	Appearance	on	on or off
Point budget	Advanced	4,000	a whole number from 100 to 20,000
Retained samples	Advanced	500,000	a whole number from 10,000 to 2,000,000

Stacked trend (`stacked-trend`)

One pane per signal, full height each, stacked on a shared time axis so a feature lines up down the whole column. What the multi-signal trend cannot do once the traces start to cross.

- **Places on:** Dashboard grid, 8×6 cells (at least 4×3)
- **Updates:** Detailed
- **History fetched when the page opens:** up to 4,000 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Signals**: Plottable, several, in order

Setting	Section	Default	Accepts
Window	Data	`1h`	a duration such as `15m` , `1h` , `24h`
Share one time axis	Appearance	on	on or off
Show grid	Appearance	on	on or off
Show the time axis	Appearance	on	on or off
Point budget	Advanced	4,000	a whole number from 100 to 20,000
Retained samples	Advanced	500,000	a whole number from 10,000 to 2,000,000

Multiplot (`multiplot`)

A board of small multiples: each signal its own pane, laid out in a grid, rows or columns. For comparing many signals side by side rather than reading one closely.

- **Places on:** Dashboard grid, 8×6 cells (at least 4×3)

- **Updates:** Detailed
- **History fetched when the page opens:** up to 4,000 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Signals**: Plottable, several, in order

Setting	Section	Default	Accepts
Window	Data	`1h`	a duration such as `15m`, `1h`, `24h`
Layout	Appearance	Grid	one of: Grid, Rows, Columns
Share the time axis	Appearance	off	on or off
Show grid	Appearance	on	on or off
Show the time axis	Appearance	on	on or off
Point budget	Advanced	4,000	a whole number from 100 to 20,000
Retained samples	Advanced	500,000	a whole number from 10,000 to 2,000,000

Thermal map (`thermal-map`)

Sensors on a surface as a colour map, face by face: absolute, or each cell's deviation from its row or face, with dead cells filled from their neighbours and marked.

- **Places on:** Dashboard grid, 8×6 cells (at least 3×3)
- **Updates:** Moderate
- **Data:**
  - **Cells, face by face in row order (faces without a tag template)**: Numeric, several, in order

Setting	Section	Default	Accepts
Faces and arrangement	Data	—	the faces in order, each with **Tag template ({row}, {col})**, **Rows** and **Columns**; the **Arrangement** and how each face is drawn. The faces, in order: each a grid of sensors named by a tag template with {row} and {col}, how it is drawn, and how the faces are arranged.
Operators may rearrange it for themselves	Behaviour	off	on or off
Decimal places	Formatting	0	a whole number from 0 to 3

Boolean track (`boolean-track`)

Boolean signals as bands, sharing the time axis above them.

- **Places on:** Dashboard grid, 12×2 cells (at least 3×1)
- **Updates:** On change
- **History fetched when the page opens:** up to 500 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Boolean signals**: Boolean, several, in order

Setting	Section	Default	Accepts
Window	Data	`1h`	a duration such as `15m`, `1h`, `24h`

State timeline (`state-timeline`)

How long the process spent in each state, in order.

- **Places on:** Dashboard grid, 12×3 cells (at least 4×2)
- **Updates:** On change
- **Data:**
  - **State source**: Entity, required

Setting	Section	Default	Accepts
Window	Data	`8h`	a duration such as `15m`, `1h`, `24h`

Execution timeline (`execution-timeline`)

Heats, batches, orders or runs laid out on a time axis.

- **Places on:** Dashboard grid, 12×4 cells (at least 4×2)
- **Updates:** On change
- **Data:**
  - **Execution source**: Entity, required

Setting	Section	Default	Accepts
Window	Data	`24h`	a duration such as `15m`, `1h`, `24h`
Show status	Appearance	on	on or off

Line account (`line-time`)

Where each machine's time went, and which one everything else was waiting for. Measured from what was recorded rather than simulated: the constraint is the machine that spends least of the window blocked or starved.

- **Places on:** Dashboard grid, 6×4 cells (at least 4×3); Process canvas, 520 × 300
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Machines, in line order	Data	—	a list, comma separated. Asset paths, upstream first — Packaging/Depalletiser, Packaging/Filler. Each machine's state signals are those paths plus the suffixes below.
Running signal suffix	Data	—	text
Blocked signal suffix	Data	—	text
Starved signal suffix	Data	—	text

Setting	Section	Default	Accepts
Faulted signal suffix	Data	—	text
Account over	Data	`8h`	a duration such as `15m`, `1h`, `24h`
Believe a state for at most	Data	`15m`	a duration such as `15m`, `1h`, `24h`
Name the constraint	Appearance	on	on or off

Run overlay (`run-overlay`)

The last runs of a type against a band built from the ones before them. Holes are drawn as holes, and every warning is shown.

- **Places on:** Dashboard grid, 8×6 cells (at least 4×3)
- **Updates:** Not live
- **Data:**
  - **Execution source**: Entity, required
  - **Signal**: Numeric, required

Setting	Section	Default	Accepts
Search window	Data	`7d`	a duration such as `15m`, `1h`, `24h`
Runs that define normal	Data	5	a whole number from 1 to 50
Recent runs to judge	Data	1	a whole number from 1 to 10
Align on	Appearance	Fraction of the run	one of: Fraction of the run, Seconds from start
Band width, in σ	Advanced	3	a number from 0.5 to 10
Show the deviation figures	Appearance	on	on or off

Statistics card (`statistics-card`)

Minimum, maximum, mean and spread over a window.

- **Places on:** Dashboard grid, 4×4 cells (at least 3×2)
- **Updates:** Not live
- **Data:**
  - **Signals**: Numeric, several, in order

Setting	Section	Default	Accepts
Window	Data	`1h`	a duration such as `15m`, `1h`, `24h`

Bar chart (`bar-chart`)

Values compared side by side.

- **Places on:** Dashboard grid, 6×4 cells (at least 3×2)
- **Updates:** On change
- **Data:**
  - **Series**: Numeric, several, in order

Setting	Section	Default	Accepts
Orientation	Appearance	Vertical	one of: Vertical, Horizontal

Distribution chart (`distribution-chart`)

How a value was distributed over a window.

- **Places on:** Dashboard grid, 6×4 cells (at least 3×2)
- **Updates:** Not live
- **History fetched when the page opens:** up to 2,000 points, never more than the project's trend point budget, counted on **Page cost**
- **Data:**
  - **Signal**: Numeric, required

Setting	Section	Default	Accepts
Window	Data	`8h`	a duration such as `15m`, `1h`, `24h`
Bins	Appearance	20	a whole number from 4 to 100

Pareto chart (`pareto-chart`)

Causes ranked by contribution, with the cumulative line.

- **Places on:** Dashboard grid, 6×4 cells (at least 3×3)
- **Updates:** On change
- **Data:**
  - **Event source**: Entity, required

Setting	Section	Default	Accepts
Window	Data	`24h`	a duration such as `15m`, `1h`, `24h`
Categories shown	Appearance	8	a whole number from 3 to 30

Point budget and retained samples

Two settings under **Advanced** decide what a chart costs:

- **Point budget** is how many points the chart asks the node for when it draws its window. It is shared among the chart's signals, not given to each: a chart of four signals with a budget of 4,000 asks for 1,000 points per signal (never fewer than 50 each). The chart clamps it to between 50 and 20,000. A **Live scope** starts at 8,000 so that its short window comes back as raw samples rather than averaged buckets.
- **Retained samples** is how many samples the browser keeps for the chart since the page opened, between 10,000 and 2,000,000. It is what lets a **Live trend**, a **Stacked trend** or a **Multiplot** hold a long run of a fast signal while it scrolls.

When the node returns fewer points than the window holds, the chart's header shows **Truncated**, and resting the pointer on it says *The server returned fewer points than the window contains*.

What the reader can do with a chart

- **Y** and **X** in the header of a **Live trend**, **Historical trend**, **Multi-signal trend**, **Data logger**, **Stacked trend** and **Multiplot** hide or show the value axis and the time axis (a **Live scope** has only **Y**: its horizontal axis is a position in the sweep, not a clock). The choice lasts while the reader looks; reopening the page brings back what you set in **Show the value axis** or **Show the time axis**.
- On a **Mini trend**, **Live trend**, **Historical trend** and **Multi-signal trend**, dragging across the chart replays just that span (*Drag across the chart to replay just that span*).
- The window badge in the header names the window the chart shows.

Context: state and **Context: run** on the live, historical and multi-signal trends shade the chart behind its traces with the states an asset was in, or the runs it was making, over the window. Bind them to a current-state or current-execution source.

Live scope, data logger, stacked trend and multiplot

These four are on the demonstration's **Trends** page (see the worked example below).

- **Live scope** draws one fast signal as an oscilloscope: the waveform against sample position, **Sweep** (redrawing from the left like a scope trace) or **Scroll**. **Points shown (0 fits the window)** at 0 shows as many samples as the window holds; a number fixes it.
- **Data logger** accumulates its signals from the moment the page opened, with the axis following the newest point. **Follow** set to **Grow to fit** keeps everything since the page opened on one axis; **Sliding window** shows only the last **Window**.
- **Stacked trend** gives each signal its own full-height pane, one above the other, on one time axis, so a step in one lines up with the others.
- **Multiplot** gives each signal its own small pane, laid out by **Layout** as a **Grid**, **Rows** or **Columns**. **Share the time axis** is off by default, since the panes are for comparing shapes side by side.

The line account

The **Line account** (`line-time`) answers where each machine's time went over a window, and which machine the rest of the line was waiting for. You name machines instead of binding signals:

- **Machines, in line order** takes asset paths, upstream first. Each machine's state signals are its path plus a suffix: `Packaging/Erector` with the running suffix `Running` reads `Packaging/Erector/Running`. A path may name a page parameter in braces, so one page can serve several lines.
- **Running signal suffix**, **Blocked signal suffix**, **Starved signal suffix** and **Faulted signal suffix** start empty in the inspector. Left unset, the component uses `Running`, `Blocked`, `Starved` and `Faulted`. Type your plant's word where it differs. Give a suffix an empty value when the plant does not report that state: its time is then shown as **Not recorded** rather than counted as something nobody measured.
- **Account over** is the window (`8h` by default). **Believe a state for at most** is how long a state signal is carried forward from its last sample before its time stops being counted (`15m`).

Each machine gets a stacked bar of **Running**, **Blocked**, **Starved**, **Faulted** and **Not recorded**, with the share of the window it spent waiting on the line beside it. With **Name the constraint** on, the machine that spent the least of the window blocked or starved is named **Constraint** above the bars, with its margin over the next machine (*clear of the next*). Below 5 % the margin is marked as narrow: *The line is balanced — the constraint is close and will move with the next changeover.*

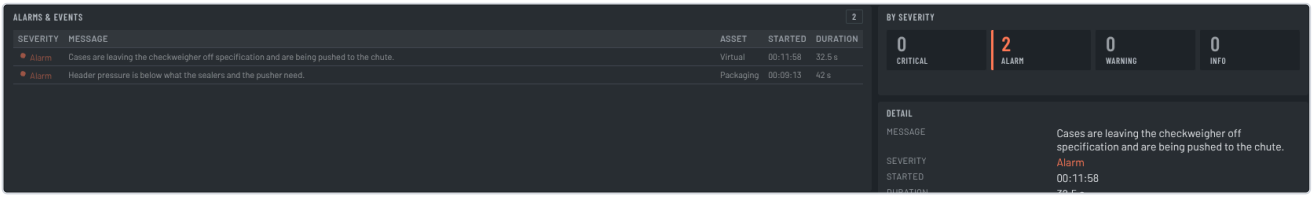
The account is about the window, not about the line in general: one account over an hour and one over a shift may name different machines. While the page is replaying, the component says *The line account is not replayed — it would report the live line's hours beside a screen showing another time. Return to live to use it.*

The run overlay

The **Run overlay** (`run-overlay`) draws the last runs of a kind against a band built from the runs before them. Bind **Execution source** to the kind of run and **Signal** to what to compare. You do not name runs: the component picks them each time it draws, from the completed runs in **Search window**: **Runs that define normal** reference runs build the band, and the **Recent runs to judge** after them are drawn against it. **Align on** chooses whether runs are compared by **Fraction of the run** or by **Seconds from start**, and **Band width, in σ** sets the band.

With **Show the deviation figures** on, a table under the chart gives each judged run its **Worst** and **Mean** deviation, the points **Outside band**, and the **Coverage** of the run the comparison could use; coverage under half is highlighted. When the comparison saw less than it appears to, the header shows **Incomplete** and the reasons are listed above the figures. With one reference run the chart says *One reference run: the line is what it did, not a range. Add more reference runs for a band.* Like the line account, it is not replayed.

Alarms and events



Alarm components on the packaging line's Event log page: the Severity counter, the Alarm list with the alarms standing now, their asset, start and duration, and the Event list below the counter

Active alarm banner (`active-alarm-banner`)

The alarm an operator must see. Mark it critical so no rule can hide it.

- **Places on:** Dashboard grid, 12×2 cells (at least 4×1)
- **Updates:** On change
- **Data:**
  - **Alarm source**: Entity, required

Setting	Section	Default	Accepts
Only what is in alarm now	Data	on	on or off
Alarms shown	Appearance	3	a whole number from 1 to 10

Alarm summary (`alarm-summary`)

Counts by severity, with the worst named.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×2)
- **Updates:** On change
- **Data:**
  - **Alarm source**: Entity, required

Setting	Section	Default	Accepts
Only what is in alarm now	Data	on	on or off

Alarm list (`alarm-list`)

Alarms, worst first, one row per condition. Read-only in this release.

- **Places on:** Dashboard grid, 8×6 cells (at least 4×3)
- **Updates:** On change
- **Data:**
  - **Alarm source**: Entity, required

Setting	Section	Default	Accepts
Rows	Appearance	20	a whole number from 5 to 200
Only what is in alarm now	Data	on	on or off
Show duration	Appearance	on	on or off
Show asset	Appearance	on	on or off

Alarm details (`alarm-details`)

The selected alarm: cause, recommended action and related signals.

- **Places on:** Dashboard grid, 4×6 cells (at least 3×3)
- **Updates:** On change
- **Data:**
  - **Alarm source**: Entity

No settings of its own.

Event list (`event-list`)

What happened, in order.

- **Places on:** Dashboard grid, 6×5 cells (at least 3×2)
- **Updates:** On change
- **Data:**
  - **Event source**: Entity, required

Setting	Section	Default	Accepts
Window	Data	`8h`	a duration such as `15m`, `1h`, `24h`
Rows	Appearance	20	a whole number from 5 to 200

Event timeline (`event-timeline`)

Events on a time axis, so clusters are visible.

- **Places on:** Dashboard grid, 12×3 cells (at least 4×2)
- **Updates:** On change
- **Data:**
  - **Event source**: Entity, required

Setting	Section	Default	Accepts
Window	Data	`8h`	a duration such as `15m`, `1h`, `24h`

Severity counter (`severity-counter`)

One count per severity, each labelled.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1)
- **Updates:** On change
- **Data:**
  - **Alarm source**: Entity, required

No settings of its own.

Severity filter (`severity-filter`)

Narrows every alarm and event component on the page to one severity. Sets the filterSeverity page parameter; All clears it.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1)
- **Updates:** Not live
- **Data:** no binding slots

No settings of its own.

Asset filter (`asset-filter`)

Narrows every alarm and event component on the page to one asset and everything under it. Sets the filterAsset page parameter; All clears it. Write a choice as path|Label; a path is taken under the root asset path.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1)
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Choices	Data	—	a list, comma separated
Root asset path	Data	—	text

How the filters narrow the page. Each filter is a row of buttons, the first of them **All**. A filter draws nothing of its own: it narrows every alarm and event component on the same page (banner, summary, list, details, event list, event timeline and severity counter), so one of each serves a whole page.

- **Severity filter** offers **All**, **Critical**, **Alarm**, **Warning** and **Info**. Pressing a severity shows only the alarms and events of exactly that severity; **All** shows every severity again. It sets the page parameter `filterSeverity` (`critical`, `error`, `warning`, `information`, or empty for all).
- **Asset filter** offers **All** and one button per entry of **Choices**, written `path|Label` (`Erector|Case erector`). A path is taken under **Root asset path**: with the root `Packaging`, `Erector` means `Packaging/Erector`; a path that already starts with the root is used as written. Pressing one shows only the alarms and events on that asset or under it, by whole path segments (`Packaging/Erector` covers `Packaging/Erector/Infeed`, not `Packaging/Erector2`); an alarm with no asset is left out while an asset is chosen. It sets the page parameter `filterAsset`. With no **Choices**, it says *No choices configured*.

The filters narrow what the components show, not what they fetch: the **Alarm source** binding still decides which alarms the page reads at all, and the shell's alarm banner and drawer are not filtered. The highlighted button is the one in force; the choice lasts until somebody presses another or leaves the page.

Process

The process components are flat symbols that change with the plant. Most go on a process canvas only; **Route diagram** is for a dashboard grid, and **Process label**, **Navigation hotspot** and **Machine model (3D)** go on either.



Process components on the packaging line's Line mimic page: a Process machine per machine with its state, a Process connector between each pair, the blocked and starved State indicators above and the buffer Vertical gauges between them

Route diagram (`route-diagram`)

The route material takes, as stations joined by arrows. Each station shows the signal beside it in the list and opens when pressed. Write a station as assetPath|Label to select an asset rather than only trend its signal.

- **Places on:** Dashboard grid, 12×3 cells (at least 4×2)
- **Updates:** On change
- **Data:**
  - **One signal per station, in route order**: Any, several, in order

Setting	Section	Default	Accepts
Stations, in route order	Data	—	a list, comma separated

Process asset (`process-asset`)

A generic asset block with its state and one headline value.

- **Places on:** Process canvas, 140 × 90
- **Updates:** On change
- **Data:**
  - **State**: Any
  - **Headline value**: Numeric

Setting	Section	Default	Accepts
Label	General	—	text
Show value	Appearance	on	on or off

Process machine (`process-machine`)

A machine outline that reflects running and faulted.

- **Places on:** Process canvas, 180 × 120
- **Updates:** On change
- **Data:**
  - **Running**: Boolean
  - **Faulted**: Boolean
  - **Headline value**: Numeric

Setting	Section	Default	Accepts
Label	General	—	text
Show value	Appearance	on	on or off

Process motor (`process-motor`)

A motor symbol with running, faulted and speed.

- **Places on:** Process canvas, 100 × 100
- **Updates:** On change
- **Data:**
  - **Running**: Boolean
  - **Faulted**: Boolean
  - **Speed**: Numeric

No settings of its own.

Process pump (`process-pump`)

A pump symbol that animates only while it is actually running.

- **Places on:** Process canvas, 100 × 100
- **Updates:** On change
- **Data:**
  - **Running**: Boolean
  - **Faulted**: Boolean
  - **Flow**: Numeric

No settings of its own.

Process valve (`process-valve`)

A valve symbol, open/closed or modulating.

- **Places on:** Process canvas, 80 × 80
- **Updates:** On change
- **Data:**
  - **Open**: Boolean
  - **Position**: Numeric
  - **Faulted**: Boolean

No settings of its own.

Process vessel (`process-vessel`)

A vessel with a level fill and its key measurements.

- **Places on:** Process canvas, 120 × 200
- **Updates:** Fast
- **Data:**
  - **Level**: Numeric
  - **Temperature**: Numeric
  - **Pressure**: Numeric
  - **Minimum**: Numeric
  - **Maximum**: Numeric

Setting	Section	Default	Accepts
Shape	Appearance	Tank	one of: Tank, Vessel, Furnace, Silo

Process sensor (`process-sensor`)

A measurement tag placed on the diagram, with quality visible.

- **Places on:** Process canvas, 120 × 50
- **Updates:** Fast
- **Data:**
  - **Actual value**: Numeric, required

Setting	Section	Default	Accepts
Unit override	Formatting	—	text
Decimal places	Formatting	1	a whole number from 0 to 9
Compact large numbers	Formatting	off	on or off
Show unit	Appearance	on	on or off

Process connector (`process-connector`)

A run of pipe or duct between two things, animated only while flow is real.

- **Places on:** Process canvas, 200 × 20
- **Updates:** On change
- **Data:**
  - **Flowing**: Boolean
  - **Flow rate**: Numeric

Setting	Section	Default	Accepts
Style	Appearance	Pipe	one of: Pipe, Duct, Electrical, Data
Animate while flowing	Appearance	on	on or off
Thickness	Appearance	6	a number from 1 to 40

Process flow arrow (`process-flow-arrow`)

Direction of flow.

- **Places on:** Process canvas, 60 × 30
- **Updates:** On change
- **Data:**
  - **Flowing**: Boolean
  - **Rate**: Numeric

No settings of its own.

Process label (`process-label`)

Static text on the diagram.

- **Places on:** Dashboard grid, 2×1 cells (at least 1×1); Process canvas, 140 × 30
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Text	General	—	text
Size	Appearance	Normal	one of: Small, Normal, Large

Navigation hotspot (`navigation-hotspot`)

An invisible or outlined region that drills into another page.

- **Places on:** Dashboard grid, 2×2 cells (at least 1×1); Process canvas, 160 × 120
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Show outline	Appearance	on	on or off
Label	General	—	text

Machine model (3D) (`model-3d`)

A 3D model of the machine, with its named parts driven by tags: a rate turns it, a position moves it, and a temperature can colour one part from its bands. Parts can also be given a colour or an uploaded texture of their own, which is what a tag then paints over. Give a part the signals that describe it and pressing it opens their charts, without leaving the page.

- **Places on:** Dashboard grid, 4×4 cells (at least 2×2); Process canvas, 280 × 240
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Model	General	—	a file from the project's assets, picked or uploaded
Part appearance	Appearance	—	a row per part: **Part**, **Paint**, **Texture** and **Tiles**
Part detail	Data	—	a row per part: **Part**, **Called**, **Signals** and **Opens on**
Show the band label	Appearance	on	on or off
Turn	Appearance	35	a number from -180 to 180

Setting	Section	Default	Accepts
Tilt	Appearance	28	a number from 0 to 89
Zoom	Appearance	1	a number from 0.2 to 5
Perspective camera	Appearance	off	on or off
Let the viewer turn it	Appearance	off	on or off
Cast shadows	Appearance	on	on or off
Label	General	—	text

Symbols that move only when the plant does

A **Process pump** and a **Process motor** draw their running mark only while **Running** is a live, good-quality *true*. A stale *true* does not animate them: the pump may have stopped since. A **Process connector** draws its flow while **Flowing** is a live *true* and **Animate while flowing** is on; with **Flowing** unbound, it flows while **Flow rate** is live and not zero. When **Faulted** is a live *true*, any process symbol carries a fault mark, and a machine or asset block says **Faulted**.

A **Process asset** or **Process machine** names its state in a word under its name: the word a bound **State** signal reports, otherwise **Running**, **Stopped** or **Faulted** from its booleans, or the quality problem when its signals are not live. A **Process vessel** fills from **Level** between **Minimum** and **Maximum** (0 and 100 when those are unbound) and writes whichever of **Temperature** and **Pressure** you bound over the fill.

Route diagrams

A **Route diagram** draws its **Stations, in route order** as boxes joined by arrows. The signals in **One signal per station, in route order** are paired with the stations by position: the first signal with the first station, and so on. A station whose signal is a boolean shows **Running** or **Stopped**, and the arrow after it is lit while it runs; a station bound to a measurement shows the reading. A station with no signal shows a dash. With no stations the diagram says *No stations configured*.

Write a station as `assetPath|Label` to show *Label* and select the asset at *assetPath* when the operator presses it. Pressing a station on the operator screen selects the asset named by the part before `|`, or by the whole entry when there is no `|`.

The machine model

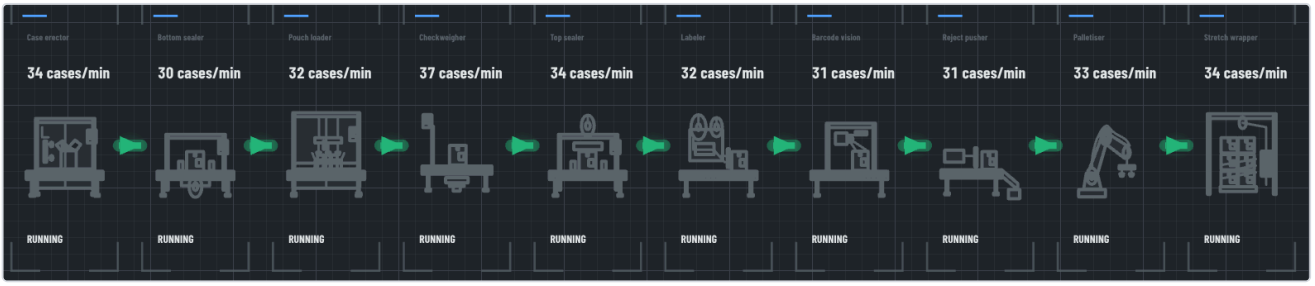
The **Machine model (3D)** draws a 3D model from the project's assets (uploading one is described in *HMI assets, expressions and parameters*) and drives its named parts from animations, listed under **Driven by**: a part turns, slides, lifts or changes colour from a tag. A part with nothing driving it does not move.

- **Part appearance** gives a part its own paint or an uploaded texture before any tag touches it; a colour animation paints over it while its band matches.
- **Part detail** names the parts that carry signals: **Part**, what it is **Called**, its **Signals**, and how much history the detail **Opens on**. Pressing such a part opens the charts of those signals without leaving the page. On the operator screen each of these parts is also a button under the model, so a keyboard user can open it too.
- **Turn**, **Tilt** and **Zoom** frame the model; **Perspective camera** is off by default, so a row of machines does not lean away at its ends. **Let the viewer turn it** lets the operator orbit it; it is off by default.
- With **Show the band label** on, every part a band has coloured is named under the model with the band's word, so the colour is never the only sign.

If the file cannot be loaded, the component says **Model unavailable** with the reason.

Shapes and symbols

The drawn shapes are the drawing tools of a process canvas (*The HMI Studio* ▸ *The canvas*). Each is inert until an animation drives it.



Shapes on the packaging line's Plant view page: a Symbol per machine coloured by its state, a Run drawn as an arrow between each pair, and Text shapes for each machine's name, rate and state

Rectangle (`shape-rectangle`)

A rectangle. With a fill-level animation it is a tank, a bunker or a bar; with a fill-colour animation it is a state block. Its outline can be drawn as corner brackets to group equipment without boxing it in.

- **Places on:** Process canvas, 160 × 100
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Fill	Appearance	Surface	a paint: any of the paints listed under *Paints*
Outline	Appearance	Outline	a paint: any of the paints listed under *Paints*
Outline width	Appearance	2	a number from 0 to 40
Outline style	Appearance	Solid	one of: Solid, Dashed, Dotted
Opacity	Appearance	1	a number from 0 to 1
Show the band label	Appearance	on	on or off
Corner radius	Appearance	0	a number from 0 to 50
Outline	Appearance	Whole box	one of: Whole box, Corner brackets, Top and bottom
Bracket length	Appearance	0	a number from 0 to 400

Ellipse (`shape-ellipse`)

An ellipse or circle.

- **Places on:** Process canvas, 110 × 110
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Fill	Appearance	Surface	a paint: any of the paints listed under *Paints*
Outline	Appearance	Outline	a paint: any of the paints listed under *Paints*
Outline width	Appearance	2	a number from 0 to 40

Setting	Section	Default	Accepts
Outline style	Appearance	Solid	one of: Solid, Dashed, Dotted
Opacity	Appearance	1	a number from 0 to 1
Show the band label	Appearance	on	on or off

Line (`shape-line`)

A straight run: a pipe, a cable, a rule. Drag its ends to place it.

- **Places on:** Process canvas, 200 × 40
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Outline	Appearance	Outline	a paint: any of the paints listed under *Paints*
Outline width	Appearance	3	a number from 0 to 40
Outline style	Appearance	Solid	one of: Solid, Dashed, Dotted
Opacity	Appearance	1	a number from 0 to 1
Show the band label	Appearance	on	on or off
Points	Advanced	`0,50 100,50`	text. Vertices as 'x,y x,y …' in a 0…100 box, so the shape stretches with its rectangle.
Kind of run	Appearance	Plain line	one of: Plain line, Pipe, Duct, Cable
Corners	Appearance	Square	one of: Square, Bent
Arrow at the start	Appearance	off	on or off
Arrow at the end	Appearance	off	on or off
Ends	Appearance	Round	one of: Square, Round

Run (`shape-polyline`)

A run with corners. Every vertex is a handle on the canvas.

- **Places on:** Process canvas, 240 × 140
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Outline	Appearance	Outline	a paint: any of the paints listed under *Paints*
Outline width	Appearance	3	a number from 0 to 40
Outline style	Appearance	Solid	one of: Solid, Dashed, Dotted
Opacity	Appearance	1	a number from 0 to 1
Show the band label	Appearance	on	on or off

Setting	Section	Default	Accepts
Points	Advanced	`0,80 35,20 70,70 100,20`	text. Vertices as 'x,y x,y …' in a 0…100 box, so the shape stretches with its rectangle.
Kind of run	Appearance	Plain line	one of: Plain line, Pipe, Duct, Cable
Corners	Appearance	Square	one of: Square, Bent
Arrow at the start	Appearance	off	on or off
Arrow at the end	Appearance	off	on or off
Ends	Appearance	Round	one of: Square, Round

Polygon (`shape-polygon`)

A closed outline of your own: a hopper, a chute, a plant boundary.

- **Places on:** Process canvas, 140 × 140
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Fill	Appearance	Surface	a paint: any of the paints listed under *Paints*
Outline	Appearance	Outline	a paint: any of the paints listed under *Paints*
Outline width	Appearance	2	a number from 0 to 40
Outline style	Appearance	Solid	one of: Solid, Dashed, Dotted
Opacity	Appearance	1	a number from 0 to 1
Show the band label	Appearance	on	on or off
Points	Advanced	`50,0 100,38 82,100 18,100 0,38`	text. Vertices as 'x,y x,y …' in a 0…100 box, so the shape stretches with its rectangle.
Corners	Appearance	Square	one of: Square, Bent

Arc (`shape-arc`)

A sector or ring segment. With a scale animation on its own arc it is a dial an author drew rather than one the catalog chose.

- **Places on:** Process canvas, 140 × 140
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Fill	Appearance	Surface	a paint: any of the paints listed under *Paints*
Outline	Appearance	Outline	a paint: any of the paints listed under *Paints*
Outline width	Appearance	2	a number from 0 to 40

Setting	Section	Default	Accepts
Outline style	Appearance	Solid	one of: Solid, Dashed, Dotted
Opacity	Appearance	1	a number from 0 to 1
Show the band label	Appearance	on	on or off
Start angle	Appearance	-120	a number from -360 to 360
End angle	Appearance	120	a number from -360 to 360
Inner radius	Appearance	0	a number from 0 to 0.95

Text (`shape-text`)

A label on the mimic. A text animation replaces its content with a live reading.

- **Places on:** Process canvas, 160 × 40
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Text	General	—	text
Fill	Appearance	Text	a paint: any of the paints listed under *Paints*
Opacity	Appearance	1	a number from 0 to 1
Show the band label	Appearance	on	on or off
Font size	Appearance	16	a number from 6 to 160
Bold	Appearance	on	on or off
Alignment	Appearance	Centre	one of: Left, Centre, Right

Symbol (`shape-symbol`)

One of the shipped equipment symbols, drawn flat and coloured by whatever drives it.

- **Places on:** Process canvas, 120 × 120
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Symbol	General	Tank	one of 117 choices
Fill	Appearance	Surface	a paint: any of the paints listed under *Paints*
Outline	Appearance	Outline	a paint: any of the paints listed under *Paints*
Outline width	Appearance	2	a number from 0 to 40
Outline style	Appearance	Solid	one of: Solid, Dashed, Dotted
Opacity	Appearance	1	a number from 0 to 1
Show the band label	Appearance	on	on or off

Icon (`shape-icon`)

One icon from the shared set, drawn as geometry so it scales and takes a semantic colour like everything else.

- **Places on:** Process canvas, 48 × 48
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Icon	General	Warning	one of the 167 icons of the shared set
Colour	Appearance	Text	a paint: any of the paints listed under *Paints* but None
Line weight	Appearance	1.7	a number from 0.5 to 6
Opacity	Appearance	1	a number from 0 to 1
Show the band label	Appearance	on	on or off

Image (`shape-image`)

A picture from the project's assets — a plant photograph, a supplier drawing — that moves, fades and blinks with a signal like any other shape.

- **Places on:** Process canvas, 180 × 120
- **Updates:** On change
- **Data:** no binding slots

Setting	Section	Default	Accepts
Image asset	General	—	a file from the project's assets, picked or uploaded
Fit	Appearance	Fit inside	one of: Fit inside, Fill and crop, Stretch
Opacity	Appearance	1	a number from 0 to 1
Show the band label	Appearance	on	on or off

The symbol library

The **Symbol** shape (`shape-symbol`) draws one of these. The Studio offers them in one list, in the Symbol tool's picker on the canvas and in the **Symbol** setting; this chapter groups them by family. Two pairs share a name in the list: **Blower** (`blower` and `blower-gas`), **Screw conveyor** (`screw` and `conveyor-screw`) and **Bucket elevator** (`elevator` and `elevator-bucket`). The id tells them apart. The packaging family is the machines of the demonstration's case-packing line.

Vessels and storage (12)

Symbol	Id
Tank	`tank`
Open bath	`tank-open`
Conical-roof tank	`tank-conical-roof`
Floating-roof tank	`tank-floating-roof`

Symbol	Id
Silo	`silo`
Hopper	`hopper`
Pressure vessel	`vessel`
Storage sphere	`sphere`
Column	`column`
Reactor	`reactor`
Drum	`drum`
Ladle	`ladle`

Pumps, fans, compressors and drives (22)

Symbol	Id
Pump	`pump`
Centrifugal pump	`pump-centrifugal`
Gear pump	`pump-gear`
Screw pump	`pump-screw`
Peristaltic pump	`pump-peristaltic`
Submersible pump	`pump-submersible`
Sump pump	`pump-sump`
Vacuum pump	`pump-vacuum`
Vertical pump	`pump-vertical`
Motor	`motor`
Electric motor	`motor-electric`
Drive	`motor-drive`
Generator	`generator`
Turbine	`turbine`
Fan	`fan`
Blower	`blower`
Blower	`blower-gas`
Compressor	`compressor`
Centrifugal compressor	`compressor-centrifugal`
Reciprocating compressor	`compressor-reciprocating`
Agitator	`agitator`
Propeller agitator	`agitator-propeller`

Valves and fittings (18)

Symbol	Id
Valve	`valve`
Control valve	`valve-control`
Gate valve	`valve-gate`
Ball valve	`valve-ball`
Globe valve	`valve-globe`
Butterfly valve	`valve-butterfly`
Check valve	`valve-check`
Plug valve	`valve-plug`
Needle valve	`valve-needle`
Motorised valve	`valve-motor`
Solenoid valve	`valve-solenoid`
Spring-return valve	`valve-spring`
Angle valve	`valve-angle`
Three-way valve	`valve-three-way`
Damper	`damper`
Flange	`flange`
Strainer	`strainer`
Reducer	`reducer`

Heat transfer and separation (16)

Symbol	Id
Heat exchanger	`heat-exchanger`
Shell-and-tube exchanger	`exchanger-shell-tube`
Plate exchanger	`exchanger-plate`
Finned-tube exchanger	`exchanger-finned`
Coil exchanger	`exchanger-coil`
Condenser	`condenser`
Electric heater	`heater-electric`
Reboiler	`reboiler`
Evaporator	`evaporator`
Boiler	`boiler`
Burner	`burner`
Stack	`stack`
Cooling tower	`cooling-tower`

Symbol	Id
Forced-draught cooling tower	`cooling-tower-forced`
Filter	`filter`
Cyclone separator	`cyclone`

Metals and heavy process (8)

Symbol	Id
Electric arc furnace	`eaf`
Caster	`caster`
Roll stand	`roll-stand`
Coil	`coil`
Kiln	`kiln`
Furnace	`furnace`
Press	`press`
Crusher	`crusher`

Handling (10)

Symbol	Id
Conveyor	`conveyor`
Belt conveyor	`conveyor-belt`
Screw conveyor	`screw`
Screw conveyor	`conveyor-screw`
Bucket elevator	`elevator`
Bucket elevator	`elevator-bucket`
Crane	`crane`
Transfer car	`car`
Weigh scale	`scale`
Robot	`robot`

Instruments and electrical (11)

Symbol	Id
Instrument	`sensor`
Field instrument	`instrument-field`
Panel instrument	`instrument-panel`
Programmable controller	`instrument-plc`
Indicator	`indicator`

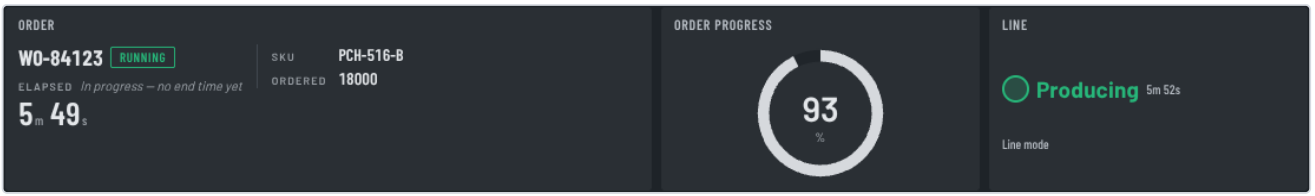
Symbol	Id
Coriolis meter	`flow-coriolis`
Rotameter	`flow-rotameter`
Turbine meter	`flow-turbine`
Venturi	`flow-venturi`
Vortex meter	`flow-vortex`
Transformer	`transformer`

Packaging (20)

Symbol	Id
Folded-case magazine	`packaging-case-magazine`
Case erector	`packaging-case-erector`
Bottom flap sealer	`packaging-bottom-sealer`
Top flap sealer	`packaging-top-sealer`
Pouch collator and loader	`packaging-pouch-loader`
Checkweigher	`packaging-checkweigher`
Print-and-apply labeler	`packaging-labeler`
Barcode and vision inspection	`packaging-vision-inspection`
Reject pusher and chute	`packaging-reject-pusher`
Articulated palletizing robot	`packaging-palletizing-robot`
Wooden pallet	`packaging-pallet`
Rotary-arm stretch wrapper	`packaging-stretch-wrapper`
Powered roller conveyor	`packaging-roller-conveyor`
Sealed flexible pouch	`packaging-pouch`
Folded RSC case blank	`packaging-case-flat`
Open RSC case	`packaging-case-open`
Sealed and labeled case	`packaging-case-closed`
Four-cup vacuum gripper	`packaging-vacuum-gripper`
Photoelectric sensor	`packaging-photoelectric-sensor`
Geared electric motor	`packaging-electric-motor`

Execution and context

These read the semantic model (runs, states, context values and relationships) rather than signals, so their slots take **Entity** sources: a current execution, a current state, an event or alarm query, or a relationship query.



Execution and context on the packaging line's Overview page: Current execution with the running order, its SKU, quantity ordered and elapsed time, and Current state reading Producing with the time in that state

Advisories (`advisory-panel`)

What the node is advising about this line — where the constraint is and what it is costing — from the advisory rules in the plant's configuration. Advisories are read, not executed: nothing here writes to a controller.

- **Places on:** Dashboard grid, 6×3 cells (at least 3×2); Process canvas, 520 × 180
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Line	Data	—	text
Show at most	Appearance	3	a whole number from 1 to 20

Current execution (`current-execution`)

Whatever the site calls a run — heat, batch, coil, order — with its identifier and how far through it is.

- **Places on:** Dashboard grid, 4×4 cells (at least 3×2)
- **Updates:** On change
- **Data:**
  - **Execution source**: Entity, required

Setting	Section	Default	Accepts
Show progress	Data	on	on or off
Show context values	Data	on	on or off
Show elapsed time	Data	on	on or off

Execution progress (`execution-progress`)

How far through the current run the process is.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1)
- **Updates:** Fast
- **Data:**
  - **Execution source**: Entity, required

No settings of its own.

Current state (`current-state`)

The state an asset is in, and for how long.

- **Places on:** Dashboard grid, 2×2 cells (at least 2×1)

- **Updates:** On change
- **Data:**
  - **State source**: Entity, required

Setting	Section	Default	Accepts
Show time in state	Data	on	on or off

Context summary (`context-summary`)

Product, recipe, crew, tooling — whatever the site attaches to a run.

- **Places on:** Dashboard grid, 4×4 cells (at least 2×2)
- **Updates:** On change
- **Data:**
  - **Context values**: Any, several, in order

No settings of its own.

Asset breadcrumb (`asset-breadcrumb`)

Where in the plant this page is, and a way back up.

- **Places on:** Dashboard grid, 6×1 cells (at least 2×1)
- **Updates:** Not live
- **Data:**
  - **Asset**: Entity, required

No settings of its own.

Recipe summary (`recipe-summary`)

The product or recipe in progress, read-only.

- **Places on:** Dashboard grid, 4×4 cells (at least 3×2)
- **Updates:** On change
- **Data:**
  - **Execution source**: Entity
  - **Context values**: Any, several, in order

No settings of its own.

Production summary (`production-summary`)

Completed runs over a window, and how they finished.

- **Places on:** Dashboard grid, 4×4 cells (at least 3×2)
- **Updates:** On change
- **Data:**
  - **Execution source**: Entity, required

Setting	Section	Default	Accepts
Window	Data	`24h`	a duration such as `15m`, `1h`, `24h`
Rows	Appearance	20	a whole number from 5 to 200

Relationship list (`relationship-list`)

What this entity is linked to, and how.

- **Places on:** Dashboard grid, 4×4 cells (at least 2×2)
- **Updates:** Not live
- **Data:**
  - **Entity**: Entity, required

No settings of its own.

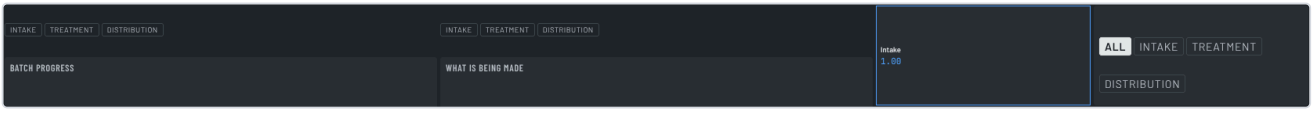
Advisories

Advisories (`advisory-panel`) has no binding slots. You name a **Line** (an asset path) and it shows what the node's advisory rules concluded about that line, most important first, up to **Show at most**. Each advisory gives the machine it is about, its margin where the rule has one, the window it covers (*over 1 h*), the sentence, and any warnings about what the evaluation could not see.

When there is nothing to show, the panel says which of these is true, because they are not the same: *Nothing to advise — no machine is clearly holding the line back.*, *No advisory rules cover this line yet.*, *This host does not evaluate advisory rules.*, *You are not permitted to read the signals these advisories are drawn from.*, or *The advisories could not be fetched.*

Where the plant's advisory rule itself names an action, the advisory carries a button with the tag and value it would write beside it. Pressing it raises the same write a **Write a value (command)** control raises, with that control's confirmation and the node's permission and command checks; the panel sends nothing itself, and offers no button in the Studio. Advisory rules are written in the plant's configuration, not on the page.

Navigation



Navigation components on the reference works' Batch page: the Area selector and Asset selector each offering Intake, Treatment and Distribution, an Asset link to Intake, and an Asset filter with All, Intake, Treatment and Distribution

Page link (`page-link`)

A button that opens another page.

- **Places on:** Dashboard grid, 2×2 cells (at least 1×1); Process canvas, 160 × 100
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Label	General	—	text
Icon	Appearance	—	text

Asset link (`asset-link`)

A link to an asset's detail page, showing its state.

- **Places on:** Dashboard grid, 2×2 cells (at least 1×1); Process canvas, 160 × 100

- **Updates:** On change
- **Data:**
  - **State**: Any

Setting	Section	Default	Accepts
Label	General	—	text

Time range selector (`time-range-selector`)

Sets the window every trend on the page reads.

- **Places on:** Dashboard grid, 6×1 cells (at least 3×1)
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Offered ranges	Data	—	a list, comma separated

Area selector (`area-selector`)

Sets the area page parameter, so one page serves several lines. Write a choice as path|Label; a path is taken under the root asset path.

- **Places on:** Dashboard grid, 4×1 cells (at least 2×1)
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Choices	Data	—	a list, comma separated
Root asset path	Data	—	text

Asset selector (`asset-selector`)

Sets the asset page parameter. Write a choice as path|Label; a path is taken under the root asset path.

- **Places on:** Dashboard grid, 4×1 cells (at least 2×1)
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Choices	Data	—	a list, comma separated
Root asset path	Data	—	text

Trend legend (`trend-legend`)

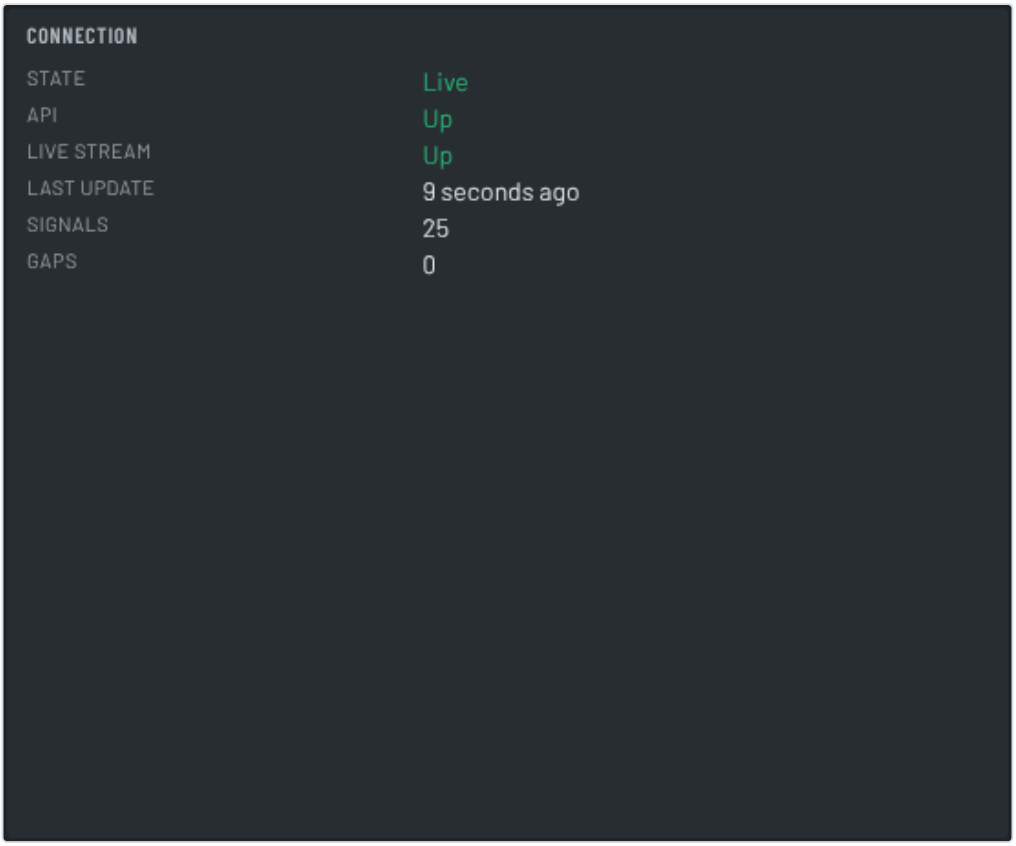
Names and current values for the signals a chart is drawing.

- **Places on:** Dashboard grid, 6×1 cells (at least 2×1)
- **Updates:** On change
- **Data:**
  - **Signals**: Plottable, several, in order

No settings of its own.

What the selectors offer. Time range selector offers the ranges in **Offered ranges**, or `15m`, `1h`, `8h` and `24h` when the list is empty, and pressing one sets the window of the page's trends. **Area selector** and **Asset selector** draw a button for each entry of **Choices**, written `path|Label` (`Intake|Intake works`), and pressing one sets the `area` or `asset` page parameter to its path. A path is taken under **Root asset path**: with the root `Reference`, `Intake` sets `Reference/Intake`; a path that already starts with the root is set as written. The button whose path is in force is highlighted. A selector with no **Choices** says *No choices configured*. **Page link** and **Asset link** do whatever their **Navigation** says; an asset link also shows the state word of its **State** binding.

Information



Connection health on the packaging line's Acquisition page: State Live, API and Live stream Up, the last update, the number of signals and no gaps

Text (`text`)

Static text. Localized with the rest of the project.

- **Places on:** Dashboard grid, 4×2 cells (at least 1×1); Process canvas, 160 × 100
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Text	General	—	text, several lines
Size	Appearance	Normal	one of: Small, Normal, Large

Image (`image`)

An image from the project's assets. A text alternative is required for accessibility.

- **Places on:** Dashboard grid, 4×4 cells (at least 1×1); Process canvas, 160 × 100
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Image	Data	—	a file from the project's assets, picked or uploaded
Text alternative	General	—	text
Fit	Appearance	Contain	one of: Contain, Cover, Stretch

Icon (`icon`)

One icon from the shared set.

- **Places on:** Dashboard grid, 1×1 cells (at least 1×1); Process canvas, 48 × 48
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Icon	General	—	text
Colour	Appearance	—	one of the theme's semantic colours

Legend (`legend`)

What the page's colours and states mean, written out.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1)
- **Updates:** Not live
- **Data:** no binding slots

No settings of its own.

Help button (`help-button`)

Opens a dialog explaining this page.

- **Places on:** Dashboard grid, 1×1 cells (at least 1×1); Process canvas, 40 × 40
- **Updates:** Not live
- **Data:** no binding slots

Setting	Section	Default	Accepts
Help text	General	—	text, several lines

Connection health (`connection-health`)

API and stream state, last update and gap count. Belongs on any page an operator relies on.

- **Places on:** Dashboard grid, 4×2 cells (at least 2×1)
- **Updates:** On change

- **Data:** no binding slots

No settings of its own.

Connection health lists **State**, **API** and **Live stream** (**Up** or **Down**), **Last update**, the number of **Signals** subscribed and the **Gaps** counted, and warns when the browser is offline. **Legend** writes out what the state colours mean: running, ready, warning, error, critical, stale, bad quality and disconnected. **Help button** draws a **?** that opens a dialog with its **Help text**.

When a page names a component this screen cannot draw

A page can name a type that the screen showing it does not have, usually because a station is running an older release than the Studio that published the page. The screen does not drop it: it draws an **Unknown component** placeholder (`unknown-component`) in its place, saying *This client cannot render '…'. Update the client, or replace the component.* with the type it could not draw. A reusable component whose definition is missing, refers to itself, or is nested too deeply is drawn the same way, with *Missing composite*, *Circular composite* or *Composite nested too deeply* and the composite's id. The placeholder is not in the palette.

Worked example: the four charts on the Trends page

The packaging demonstration's **Trends** page holds one of each of the four plotted charts that are not ordinary trends, and under them a multi-signal trend that uses the two trace drawings an ordinary trend can be given. Open it in the Studio to follow along.



The Trends page: Pouch loader rate — live, Machine rates — since open, Stacked — rate, OEE, buffer, Multiplot — the line, and Case weight against specification

1. **Pouch loader rate — live** is a **Live scope** with **Signal** bound to `Virtual/Computed/LoaderThroughput` and **Window** `30s`. Its header has only the **Y** button and a `30 s` badge. Use a scope like this for a signal whose shape over seconds is the point.

2. **Machine rates — since open** is a **Data logger** with six throughput signals in **Signals**, each given a **Label** on its binding (*Case erector*, *Pouch collator* …), which is the name its legend shows. **Window** is `15m` and **Follow** is **Grow to fit**, so after a commissioning run you can see the whole ramp since the page opened.

3. **Stacked — rate, OEE, buffer** is a **Stacked trend** of the loader throughput, the OEE and the buffer between the loader and the checkweigher, over `1h`. The three have different units, so each gets its own pane and its own value axis, with one time axis under them.

4. **Multiplot — the line** is a **Multiplot** of four machines' throughput, with **Layout** at **Grid**. Each pane has its own scale, so a machine running slowly shows its own shape rather than a flat line under a faster one.

5. **Case weight against specification** is a **Multi-signal trend** of `Packaging/Checkweigher/CaseWeight` and `Packaging/Checkweigher/WeightTarget`, with **Share one value axis** on and **Run the axis past now by (minutes)** at 3. Select the weight's binding and open **How it is drawn**: **Drawn as** is **A point at each reading**, so each case's weight is a mark and nothing is drawn between two cases. The target's binding is **A profile, carried past now**: it is dashed, and after the now line it carries on at its latest value to the end of the axis — the specification the next cases will be weighed against.

To try a budget: select the stacked trend, open **Advanced** and look at **Point budget** (4,000). Shared among three signals, it asks for about 1,333 points per signal. The **Page cost** tab shows what the whole page asks for.

Worked example: the control-room panels on the Overview and Machines pages

The demonstration's **Overview** and **Machines** pages are laid out in the control-room style: readings packed into panels rather than one tile each, and each of the three panel components below placed where the plant has a use for it.



The Overview page: OEE with its three factors as tiles, line rate, order progress and alarm counts; machine rates as a panel of tiles beside the shift's counts; the rate trend with its target carried past now beside deviation from aim; the case cycle as a step bar; the order, the line state and the active alarms

1. **OEE** is a **KPI card** whose **Readings it is made of, as tiles under it** list holds `Virtual/Computed/Availability`, `Virtual/Computed/Performance` and `Virtual/Computed/Quality`, labelled `availability`, `performance` and `quality`: the three factors are drawn as tiles under the headline, in the text ink whatever band the headline is in. **Show trend** is off here — at three rows the factors are the better use of the card, and the OEE page carries the trend.
2. **Machine rates** is a **Value grid** of the six machines' throughput, **Columns** 3. Each binding has a **Label** — `caseErector`, `pouchCollator` … — which is a key in the project's string table, so the tile says *Case erector* in English and *Formadora de cajas* in Spanish. A label that is not a key is shown as written.
3. **Deviation from aim** is a **Deviation from aim** panel. **Readings** holds six signals — the loader's throughput, the case weight, two seal temperatures, the bottom seal pressure and the air header, each labelled with a string-table key (`lineRate`, `caseWeight`, `sealBottom` …) so the row reads in the viewer's language — and **Aims** holds six in the same order: the line's target rate, the weight specification, and four **Constant** bindings for the aims no tag carries (178, 181, 4.4 and 6.3). **Deviation as a percentage of the aim** is on, so readings in °C, g and bar share one scale; **In range within ± this of the aim** is 2.5, which is where OK turns into HIGH or LOW; and **Bar full scale** is 10.
4. **Case cycle** is a **Step progress** bar. **Current step** is bound to `Packaging/CasePacking/CaseState`, which reports a word — *erecting*, *pouch-loading* … — and **Steps, in order** lists each as `value|Label|share`, for example *pouch-loading*|*step.load*|*6*: the value the signal reports, a string-table key for the name, and the seconds the step takes. **Done so far** is `CycleTimeSec` and **Total** a constant 78, so the now line sits where the cell is in its 78-second cycle and the caption reads *Cycle 41 / 78 s*.
5. **Unaccounted** and **Air pressure** are **Numeric value** tiles with a caption line: **Caption under the reading** is a key (`erected`, `alarmBelow`) and **Caption value** is bound — to the infeed count, and to a constant 5.6 for the alarm limit.

6. On the **Machines** page, **Machines** is a **Key-value table** with **Column headings** `throughput, rated` and **Row labels** naming the ten machines: twenty readings filling the table two to a row. **Rate against rated** is a second deviation panel, each machine's throughput against its own rated speed, with a tolerance of 8 %. Neither shows a machine's raw speed (`…/Rate`), which runs on through a fault or a wait; throughput is nothing while the machine is not producing.

Worked example: machines and conveyors on the Line mimic page

The **Line mimic** page draws the case-packing line as ten **Process machine** components joined by nine **Process connector** components, with state indicators and buffer gauges around them.

1. Select **Case erector**. Its **Data** has **Running** bound to `Packaging/Erector/Running` and **Faulted** to `Packaging/Erector/Faulted`; **Headline value** is empty, so the box shows its name and its state word only.
2. Open **Navigation**: the action is an **OpenCard** of the `machine` page with the parameter `machine=Erector`. Pressing the erector on the operator screen opens the **Machine** page as a card, showing the erector.
3. Select the connector after the erector (the inspector names it *erector-conv*). **Style** is **Pipe**, **Thickness** is 8 and **Animate while flowing** is on. **Flowing** is bound to `Packaging/BottomSealer/Running`: the connector shows flow while the machine it feeds is running, not merely while the erector is.

To add an eleventh machine, drop a **Process machine** and a **Process connector** from the **Process** category, bind them the same way, and give the machine the same **OpenCard** navigation with its own `machine` value.

Worked example: a line account on the OEE page

The demonstration's **OEE** page shows **Blocked** and **Starved** as two **Boolean track** components, one band per machine, and the **Where the constraint is** advisories. A **Line account** puts the same question into one component that names the constraint over the window itself.



The OEE page: the four OEE cards, The three factors trend, Blocked and Starved tracks per machine, Where the constraint is, and Shift to date

1. In the Studio, open the **OEE** page and drop a **Line account** from **Trends and analytics** onto the grid (it takes 6×4 cells; it can be no smaller than 4×3).
2. In **Data ▸ Machines, in line order**, enter the machines upstream first: `Packaging/Erector`, `Packaging/Loader`, `Packaging/Checkweigher`, `Packaging/Labeler`, `Packaging/Vision`, `Packaging/Palletiser`. These are the machines the **Blocked** and **Starved** tracks read.
3. Leave the four suffixes unset. The demonstration's signals are named `…/Running`, `…/Blocked`, `…/Starved` and `…/Faulted`, which are the suffixes the component uses when none is set.
4. Set **Account over** to `1h` to match the tracks beside it, and leave **Believe a state for at most** at `15m`.
5. Use **Preview**. Each machine has a bar, and the constraint is named above them with its margin. Compare it with the advisory panel: both are measured from the same recorded signals, but the advisory speaks only when its rule's margin and confirmations are met, while the account always names the machine for its own window.

Save the draft and publish as described in *The HMI Studio ▸ Publishing, versions and rollback*.

HMI assets, expressions, parameters and keys

> **Module: Web HMI.** This chapter covers four things the HMI Studio and the operator screens share: the pictures and 3D models a page shows, the expression language a binding can use, page parameters, and every keyboard shortcut. It is only present on installations licensed for the Web HMI module.

This chapter is the reference half of the HMI. The *The HMI Studio* chapter tells you how to draw, bind and publish a page; the *The browser operator HMI* chapter tells you what an operator sees; the *HMI component reference* chapter describes each component. This chapter explains the parts those three lean on and do not explain in full:

- **Assets.** Images and 3D models you upload, what each file type can and cannot do, the size limits, where the files live and how they travel between installations.
- **Expressions.** The small formula language behind the **Expression** binding source: syntax, operators, functions and the messages you get when an expression is wrong.
- **Page parameters.** How one page serves many machines, how a value reaches a page, and the **Tabs** component that switches what a page shows.
- **Keys.** Every shortcut of the Studio and of the operator screens.

The examples use the packaging demonstration (`examples/demo/packaging.meelmohmi`), the same line every other chapter uses.

Assets: images and 3D models

An asset is a file a page shows: a logo, a site plan, a photograph of a panel, a texture for a machine, or a 3D model of the machine itself. You upload it once and then pick it in any page of the project.

How an asset is identified

Every asset is named by the SHA-256 hash of its bytes, written as 64 lowercase hexadecimal characters. You never type this name, because the Studio lists assets by file name, but it explains three behaviours you will notice:

- **A published page shows exactly the bytes it was published with.** Nobody can replace the drawing under a published page without the page itself changing, because a different file is a different asset.
- **Uploading the same file twice gives one asset.** The library does not fill up with copies of one logo. Uploading the same bytes on another installation gives the same name there, which is what makes missing media on import easy to repair (see Moving a project: export and import).
- **An asset never changes once stored.** Browsers keep it in their cache for a year, so a large machine model is downloaded once per panel, not every time the page reloads.

What you can upload

The file type is decided from the file's extension, never from its content, and checked against a fixed list. Anything else is refused.

Kind	Extensions	Stored as	Largest file
Image	`.png`	`image/png`	4 MB

Kind	Extensions	Stored as	Largest file
Image	`.jpg`, `.jpeg`	`image/jpeg`	4 MB
Image	`.webp`	`image/webp`	4 MB
Image	`.gif`	`image/gif`	4 MB
Image	`.svg`	`image/svg+xml`	4 MB
3D model	`.glb`	`model/gltf-binary`	24 MB
3D model	`.gltf`	`model/gltf+json`	24 MB
3D model	`.obj`	`model/obj`	96 MB
3D model	`.stl`	`model/stl`	96 MB

The limits are binary megabytes (4 MB is 4,194,304 bytes). OBJ and STL get four times the glTF limit because they store geometry as text, which takes about four times the space of the same machine in a `.glb`. Both limits admit roughly the same decimated machine; an unreduced CAD assembly exceeds either.

An empty file is refused. An SVG is served with headers that stop it running script, so an uploaded SVG is safe to show on a page.

Which 3D format to use

Ask for **`.glb`** whenever you can. Every CAD package exports glTF, and a `.glb` packs the geometry, materials, textures and animation clips into one file. A `.gltf` that refers to separate `.bin` or image files is accepted as a single file, but the asset store cannot resolve the files it refers to, so export the self-contained `.glb` form instead.

OBJ and STL are accepted because they are what machine builders usually hand over. What each format can do on a page:

Can the page…	`.glb` / `.gltf`	`.obj`	`.stl`
Bind a tag to one named part (turn it, move it, colour it)	Yes, by the node names in the file	Yes, by its `o` and `g` names	No. The file has no parts; the whole model is one part called `model`
Move or turn the whole machine from a tag	Yes	Yes	Yes, through the part `model`
Play an animation clip from a tag	Yes, if the file has clips	No, OBJ carries no animation	No
Use the file's own materials and colours	Yes	No. The `.mtl` file an OBJ names cannot be resolved	No. STL has no materials
Paint or texture a part with **Part appearance**	Yes	Yes	Only the whole model

An OBJ or STL is drawn in the theme's own surface colour. For an OBJ, every name on an `o` or `g` line becomes a part, and a `g` line naming several groups (`g left right`) gives one part per name. An unnamed `g` line names nothing.

The Studio reads the part and clip names out of the file for you, using the same reading the publish check uses, so the **Part** lists in the animation, **Part appearance** and **Part detail** editors offer only names the file has. If that reading fails, the field becomes a text box and you can type the name.

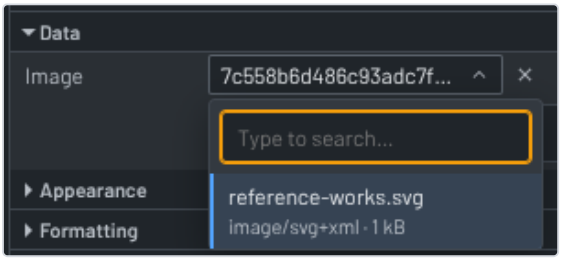
Uploading from the inspector

There is no separate asset library screen. You upload where you need the file: in any media field of the inspector.

Where	Field	Component or editor	Can upload here
Inspector, **General**	**Image asset**	**Image** (the shape drawn with the `M` tool, freeform pages)	Yes
Inspector, **Data**	**Image**	**Image** (information component, grid and freeform)	Yes
Inspector, **General**	**Model**	**Machine model (3D)**	Yes
Inspector, **Part appearance**	**Texture**	**Machine model (3D)**, one row per part	No. Lists uploaded images only
Page settings (freeform page)	**Background picture**	The page's canvas	No. Lists uploaded assets

A media field with upload has two parts:

Part	What it does
The picker	Lists the assets of this project, plus any shared by every project. Each entry shows the file name, and the type and size, such as `model/gltf-binary · 8001 kB`. While nothing is uploaded it says *nothing uploaded yet*; otherwise *choose an image or model*. Type to search; **Clear** removes the choice.
The file chooser	Opens your computer's file dialog, filtered to every allowed extension. The chosen file is uploaded at once, the field says *Uploading packaging-line.glb...*, and when it is stored the field points at the new asset.



The Image field in the Data section of the reference works' Image component, after reference-works.svg was uploaded with the file chooser under it: the picker names the stored asset by its id, and its list shows reference-works.svg, image/svg+xml · 1 kB

The file chooser offers every allowed extension in every media field, so choose the kind of file the field is for: an image for **Image asset** and **Image**, a model for **Model**.

What can go wrong, and what the field says:

Message under the field	Cause	What to do
case.dwg is not a kind of file this store takes.	The extension is not in the table above	Export to one of the listed types
line.glb is 31 MB; the limit is 24 MB.	The file is larger than its type's limit. Checked before anything is sent	Decimate the model, or compress the image
The upload exceeds the 24 MB limit for model/gltf-binary.	The same check, made by the server while the file arrives	As above

Message under the field	Cause	What to do
The upload was empty.	A zero-byte file	Export the file again

To use a texture or a background picture, first upload the image through an **Image** or **Image asset** field (on any page of the project). It then appears in the **Texture** and **Background picture** lists. **Background picture** appears in **Page settings** only once the project has at least one asset; choose **— no picture —** to remove it.

For **Part appearance**, press **Give a part its own appearance** to add a row, then set:

Field	Meaning
Part	The part the row is about, from the model's own names, or **The whole machine**
Paint	A colour from the theme's vocabulary, or **— leave it as it is —**
Texture	An uploaded image wrapped around the part. Says *no images uploaded yet* when there are none
Tiles	How many times the texture repeats across the part (at least 0.1)

A tag-driven fill animation paints over this appearance while its band matches, and the part returns to it when the band stops matching.

Checks at publish

The publish check refuses a page whose assets would leave a hole on an operator's screen:

Message	Meaning
'Model' names asset '...', which is not in this installation's library.	The field names an asset this installation does not have. Advice: *Upload it, or pick one that is already there.*
This model has no part named 'Arm_2'.	An animation names a part the file does not contain. The message lists the parts it does have, or says *The model names none of its parts, so nothing in it can be bound. Export it with names.*
This model has no animation called 'Pick'.	An animation names a clip the file does not contain. Always the case for OBJ and STL

Who may upload, and where the files are

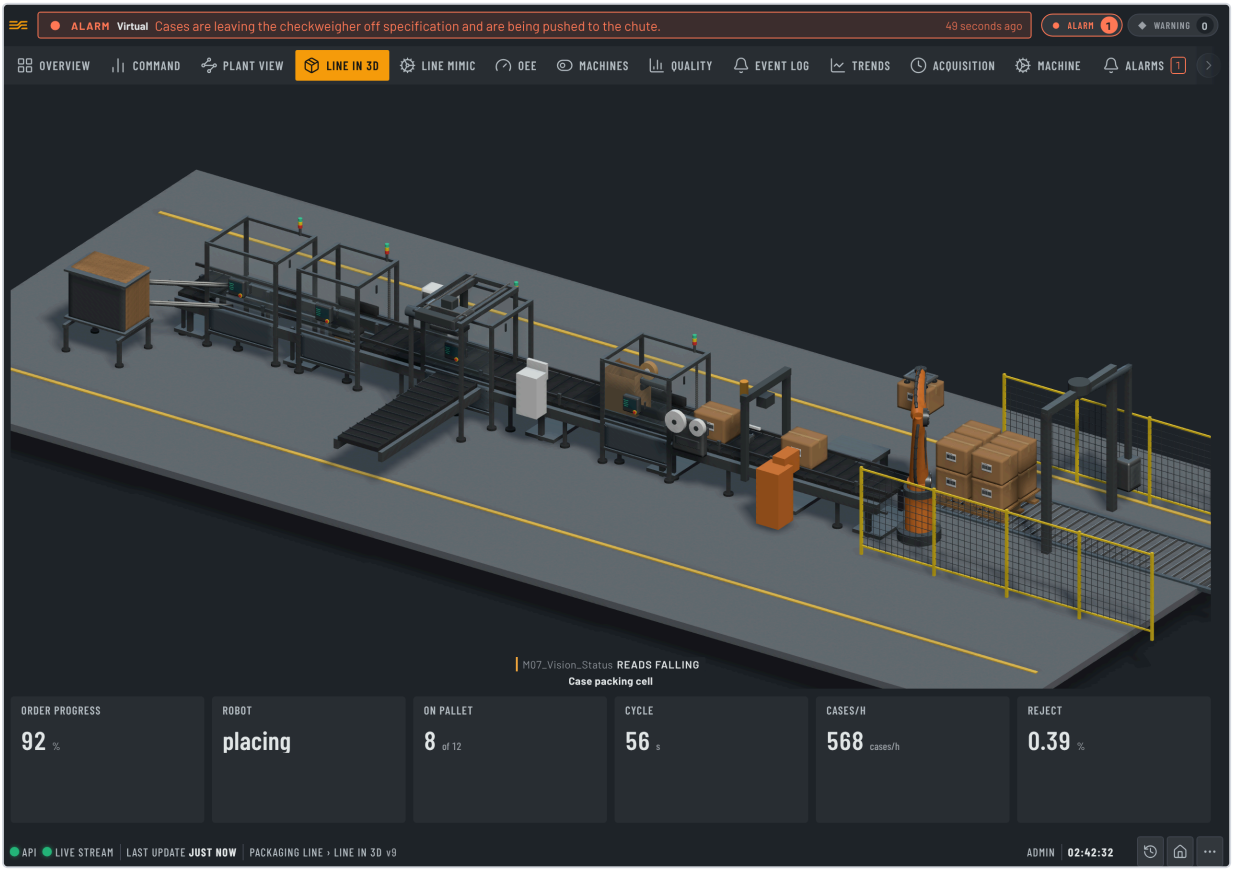
Action	Permission
Upload, list, pick, delete	**Edit pages** (`Hmi.EditPages`)
See an asset on a published page	**View pages** (`Hmi.ViewPages`)

Assets are stored on the node, in the `assets` folder of the HMI directory: by default `hmi/assets` under the web host's state directory, or under the directory named by the `MEELMO_HMI` environment variable. Each asset is two files, `{id}.bin` with the bytes and `{id}.json` with the file name, type, size, upload time, uploader and project. An upload is written to a temporary file and moved into place, so a host stopped mid-upload leaves no half-written asset. Uploads and deletions are recorded in the audit trail as `hmi.asset.uploaded` and `hmi.asset.deleted` .

The Studio has no delete button for assets. An asset can be removed only through the API (`DELETE /api/v1/hmi/assets/{id}` , with **Edit pages**), and that call does not check whether a published page still uses the asset. A page that loses its asset shows an empty space, so remove assets only when you know nothing uses them.

Worked example: the packaging line in 3D

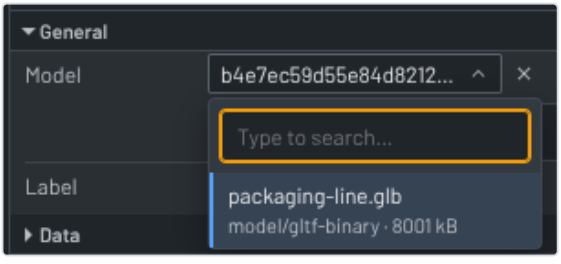
The **Line in 3D** page of the packaging demonstration shows `packaging-line.glb`, a 7.8 MB glTF of the whole case-packing cell, in a **Machine model (3D)** component labelled *Case packing cell*.



The Line in 3D page: the packaging cell drawn from packaging-line.glb, with a status label for the checkweigher display and six value tiles under the model

To put the same model on a page of your own:

1. Open **Studio**, open the **Packaging line** project, and open the page that will hold the model (a grid page or a freeform page; both take the component).
2. In **Components**, search `model` and drag **Machine model (3D)** onto the page.
3. With it selected, find **Model** in the inspector's **General** section. Use the file chooser under the picker to upload `examples/demo/packaging-line.glb` from the installation's demonstration files. The field reads *Uploading packaging-line.glb…* and then names the stored asset by its id; its list shows the file name, type and size. Because the bytes are the demonstration's own, the asset gets the same name the demonstration page uses, and both pages share one stored file.



The Model field of the Line in 3D machine model after the upload: the picker shows the asset's id, and its list shows packaging-line.glb, model/gltf-binary · 8001 kB

4. Under **Part appearance**, press **Give a part its own appearance**, choose the part `Safety_Aisles` and set **Paint** to **Attention**, as the demonstration does. The part list comes from the file; you do not type it.
5. In **Appearance**, set **Turn** to `34`, **Tilt** to `26` and **Zoom** to `1.45`, and switch on **Let the viewer turn it** if operators may spin the model.
6. Press **Validate**. A missing part or clip name is an error here, before anything is published.

Expressions

An expression computes a value for display from the other bindings of the same component. It exists so a card can show, for example, actual against target without somebody creating a virtual signal for a number nobody needs recorded.

Expressions are **presentation only**. Nothing an expression produces is recorded, alarmed, or visible to any other screen, report or client. If a value must be kept or alarmed, make it a virtual signal instead (see the *Signals: the signal tree, virtual and computed signals* chapter).

Where you write one

1. Select the component and open the inspector's **Data** section.
2. In the slot that should show the computed value, open the source list (its accessible name is *What Actual value is bound to*) and choose **Expression**. Only slots that take numbers or true/false values offer it.
3. Type the expression in the **Expression** box. Its placeholder is `actual - target`.

What the box shows	Meaning
Reads: target, trend (help under the box)	The expression parsed. These are the slots it reads
This expression cannot be read. followed by the reason	It did not parse. The reason says where (see the table of messages below)
Functions: abs, avg, ceil, … (hint under the box)	Every function you may call



The Actual value slot of a KPI card on the Overview page with its source set to Expression: the box holds percent(trend, target), the help under it reads Reads: trend, target, and the hint lists every function

What an expression can name

An expression can name exactly two kinds of thing: **the binding slots of its own component**, and **the functions in the table below**. There are no signal ids, no page parameters, no other components, no dotted paths and no variables. To use a signal, bind it to another slot of the same component and name that slot.

A slot is named by its internal name, which is a single word:

Slot shown in the inspector	Name in an expression
Actual value	`actual`
Target value	`target`
Predicted value	`predicted`
Trend signal	`trend`

Each component's slot names are in the *HMI component reference* chapter. If you name a slot the component does not have, the publish check lists the ones it does have.

Slot names are not case sensitive (`Target` and `target` are the same slot). Function names are: `abs(x)` works, `ABS(x)` is refused.

Syntax

Element	Rules	Examples
Number	Digits with an optional decimal point, always `.` . No exponent, no thousands separator, no sign of its own (a leading `-` is the minus operator)	`100` , `0.5` , `.5`
Slot name	Starts with a letter or `_` ; then letters, digits or `_`	`actual` , `target`
Function call	Name, `(` , arguments separated by `,` , `)`	`round(actual, 1)`
Parentheses	Group a sub-expression	`(actual - target) / target`
Conditional	`condition ? when-true : when-false`	`actual > target ? actual : target`
Spaces	Anywhere between elements; ignored	

There are no text values and no `true` or `false` words. A comparison gives `1` for true and `0` for false, and any value other than `0` counts as true.

Operators

From the loosest binding to the tightest. Operators on the same row are evaluated left to right.

Precedence	Operators	Meaning	Result
1 (loosest)	`? :`	Conditional	The chosen branch
2	`||`	Or	`1` if either side is non-zero, else `0`
3	`&&`	And	`1` if both sides are non-zero, else `0`
4	`==` `!=`	Equal, not equal (within 0.000000001)	`1` or `0`
5	`<` `<=` `>` `>=`	Comparison	`1` or `0`
6	`+` `-`	Add, subtract	Number
7	`*` `/` `%`	Multiply, divide, remainder	Number; no value when dividing by `0`

Precedence	Operators	Meaning	Result
8 (tightest)	`-` `+` `!` (before a value)	Negate, plus, not	`!x` is `1` when `x` is `0`, else `0`

A single `=` is not an operator; write `==`.

Functions

Seventeen functions. Arguments may themselves be expressions.

Function	Arguments	Returns
`abs(x)`	1	The absolute value
`round(x)`	1	`x` rounded to a whole number, halves away from zero
`round(x, n)`	2	`x` rounded to `n` decimals (`n` taken as 0 to 15)
`floor(x)`	1	The largest whole number not above `x`
`ceil(x)`	1	The smallest whole number not below `x`
`min(a, b, …)`	1 to 8	The smallest
`max(a, b, …)`	1 to 8	The largest
`clamp(x, low, high)`	3	`x` held between `low` and `high`. Keep `low` at or below `high`
`sqrt(x)`	1	The square root; no value when `x` is negative
`pow(x, y)`	2	`x` to the power `y`
`avg(a, b, …)`	1 to 8	The mean
`sum(a, b, …)`	1 to 8	The total
`percent(part, whole)`	2	`part / whole × 100`; no value when `whole` is `0`
`deviation(actual, reference)`	2	`actual − reference`
`sign(x)`	1	`-1`, `0` or `1`
`coalesce(a, b, …)`	1 to 8	The first argument that has a value
`isNull(x)`	1	`1` if `x` has no value, else `0`
`formatDuration(x)`	1	`x` unchanged, as the component's value. To show seconds as *2h 14m*, set **Formatting ▸ Show as** to **A duration, from seconds**

When there is no value

A slot whose signal has not arrived, is not bound, or is the expression's own slot has **no value**. The expression language never turns "no value" into zero, because `actual - predicted` with a missing prediction would otherwise show a confident wrong number.

Case	Result
Any operator with an operand that has no value	No value
Any function except `coalesce` and `isNull`, with an argument that has no value	No value

Case	Result
`coalesce`	Skips arguments with no value
`isNull`	`1` for no value
A conditional whose condition has no value	The *when-false* branch
Division or remainder by `0`, `sqrt` of a negative, `percent` of a zero whole	No value

A component shows "no value" with its **Text when there is no value** (a dash unless you change it).

The result's quality follows its inputs. If every input is good, the result is good (or *no data* when the expression produced no value). Otherwise it takes the worst state of its inputs, and it is stale if any input is stale. Its time is the oldest input's time. A threshold band on the component therefore never lights from a value computed out of a reading that has gone stale.

The unit of the result is not taken from the inputs. Give it one in **Formatting ▸ Unit override**.

Limits

Limit	Value
Length	512 characters
Nesting	24 levels
Size	256 terms (numbers, names, operators and calls)
Arguments	8 for the functions that take a list

The limits are checked when the expression is parsed, so an expression that breaks one is refused at publish rather than slowing a screen.

Messages

While you type, the **Expression** box shows the parser's sentence. At publish, the validation strip shows the same sentence inside '*actual*' *has an expression that cannot be parsed: …*, or one of the two binding messages.

Message	Cause
An expression cannot be empty.	The box is empty or blank
An expression may be at most 512 characters; this one is 530.	Too long
The expression ends where a value was expected.	It ends after an operator, as in `actual -`
A '(' is not closed.	Unbalanced parentheses
A conditional needs a ':' between its two results.	`a ? b` without `: c`
'*#*' *is not something an expression may contain.*	A character outside the grammar, including quotes, `[`, `.` before a letter, `=` alone
'*1.2.3*' *is not a number.*	A malformed number
'*avrg*' *is not a function an expression may call.*	Unknown or wrongly capitalised function

Message	Cause
The call to 'max' is not closed.	Missing `)`
The call to 'max' has a stray ';'.	A separator other than `,`
'clamp' takes 3 arguments; this call passes 2.	Wrong number of arguments
The expression has trailing text: ')'.	Something after a complete expression
An expression may nest at most 24 levels deep.	Too deep
An expression may contain at most 256 terms.	Too large
'actual' reads 'speed', which is not a slot on 'kpi-card'.	A name that is not one of this component's slots. The remedy lists them: *Available slots: actual, target, trend.*
'actual' is an expression binding with no expression.	**Expression** chosen, box left empty

If a page somehow reaches an operator with an expression that does not parse, the component shows *This expression could not be parsed.* instead of a value.

Examples

Purpose	Expression
Gap to target	`actual - target` or `deviation(actual, target)`
Speed as a percentage of rating	`percent(trend, target)`
Gap as a percentage, one decimal	`round((actual - target) / target * 100, 1)`
The larger of two readings	`max(actual, target)`
A prediction, or the actual value while there is none	`coalesce(predicted, actual)`
`1` while over target, else `0`	`actual > target`
Hold a value inside its scale	`clamp(actual, 0, 100)`
Show zero instead of a negative	`actual < 0 ? 0 : actual`

Worked example: checkweigher speed against its rating

The checkweigher's throughput `Virtual/Computed/CheckweigherThroughput` and its rated speed `Packaging/Checkweigher/RatedSpeed` are both in the demonstration. This example adds a card showing throughput as a percentage of rating.

1. Open the **Overview** page of the **Packaging line** project in the Studio.
2. Drag a **Kpi card** into a free area of the grid.
3. In **Data**, drag `Virtual/Computed/CheckweigherThroughput` from the **Signals** tab onto **Trend signal**, and `Packaging/Checkweigher/RatedSpeed` onto **Target value**.
4. In **Actual value**, choose **Expression** in the source list and type `percent(trend, target)`. The help under the box reads *Reads: trend, target*.
5. In **Formatting**, set **Unit override** to `%` and **Decimal places** to `0`. Switch off **Show change against target**: the target is in cases per minute and the value is a percentage, so a difference between them means nothing.

6. In **Thresholds**, add a band `LessThan` `80`, colour *Warning*, label `Below rating`.
7. In **General**, type `Checkweigher speed` in the English box of **Title**.
8. Press **Preview**: the card shows a number near `100 %`, turning to **Below rating** when the checkweigher slows. Press **Validate**, then **Save draft** and **Publish…**.

Page parameters

A page parameter is a named text value a page carries while it is shown, such as `machine=Loader`. Bindings, visibility rules and interactions on the page can use it, so a single page can serve ten machines. The packaging demonstration's **Machine** page is one page for all ten machines of the line.

Declaring a parameter

A page lists the parameters it accepts:

Field	Meaning
Name	Letters, digits, `-` and `_`. Unique on the page
Display name	A localized name for the parameter
Default value	Used when the page is shown without the parameter
Required	The page needs it. A required parameter may not also have a default
Kind	A hint: text, asset path, signal id, number or duration

The Studio inspector has no editor for this list in this version. A page gets its parameters from where it came from: a project bundle you import (see Moving a project: export and import), or a template saved from a page that declares them (**New page** keeps the template page's parameters). The demonstration's **Machine** page declares one parameter, `machine`, with default `Loader`.

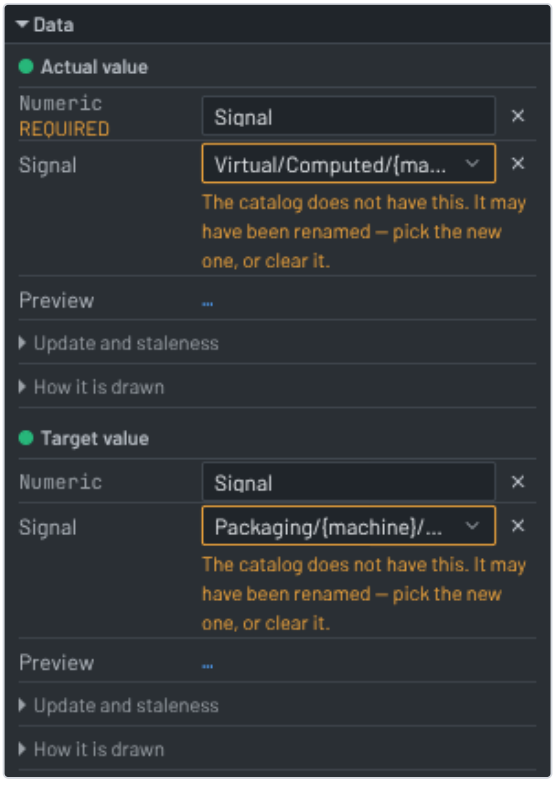
The publish check enforces the declarations:

Message	Cause
A page parameter must have a name.	An empty name
'line 1' is not a usable parameter name: use letters, digits, '-' and '_'.	A bad name
Parameter 'machine' is required and also has a default, which makes the requirement meaningless.	Both set
Page parameter 'machine' is declared more than once.	A duplicate
'value' references the page parameter 'machine', which this page does not declare.	A `{machine}` in a binding's signal or asset path on a page that does not declare `machine`
'value' reads the page parameter 'machine', which this page does not declare.	A **Page parameter** binding naming an undeclared parameter

Using a parameter on the page

Where	How	Example from the Machine page
A signal id or asset path in a binding	Write the parameter's name in braces. It is replaced when the page is shown	`Packaging/{machine}/RatedSpeed` becomes `Packaging/Loader/RatedSpeed`

Where	How	Example from the Machine page
A slot's source	Choose **Page parameter** and pick the declared name in **Parameter**. The slot shows the parameter's text, or *no data* when it is absent	A heading showing which machine this is
A visibility rule	**Page parameter equals** with **Parameter** and **Value** (compared ignoring case), or **Page parameter is present** with **Parameter** (present and not blank)	Show a panel only for `machine` = `Reject`
An interaction's **Parameters**	Write `{name}` in a value to pass the current value on	`machine={machine}`
An asset path in a visibility rule (**An execution is open**, **Asset is in a state**, **Alarm count above**)	Braces, as in bindings	`Packaging/{machine}`



The Data section of the Actual target on the Machine page: Actual value bound to Virtual/Computed/{machine}… and Target value to Packaging/{machine}/…. The inspector says The catalog does not have this under each, because an id with a parameter in it names no catalogue signal until the page is shown

A `{name}` that the shown page does not have is left in place, braces and all. A signal id that silently lost a segment would name a different signal; one that keeps its braces names nothing and shows as missing. Because a parameterised id cannot be checked against the catalogue at publish, check it in **Preview**.

The signal pickers offer catalogue signals only and cannot enter `{machine}`. To create another parameterised binding, copy or duplicate a component that already has one (`Ctrl` + `C`, `Ctrl` + `V`, or `Ctrl` + `D`) and change what differs.

A **Page parameter** binding gives text, not a number, so an expression that names such a slot reads no value.

How a parameter gets its value

Way	Sets	Notes
The page's declared default	The parameter, when nothing else has	Applied each time the page is shown
Open a page, with **Parameters**	Any parameters, on the page opened	The page starts with the current page's parameters, and the listed ones override them, as for a card. They are also written in the address
The address, as in `/page/machine?machine=Loader`	Any parameters, on that page	Read when the page is opened from the address: a bookmark, a typed address, or the browser's back and forward
Show a page as a card or **Show a page as a dialog**, with **Parameters**	Any parameters, for the card or dialog	The card starts with the current page's parameters, and the listed ones override them. This is how the demonstration passes `machine`
Set a page parameter, with **Parameters**	Any parameters, on the current page	The page re-plans its data at once
Select an asset or **Open an asset**, with **Asset path**	`asset`	
Select an area, with **Asset path**	`area`	
Set the time range, with **Range**	`range`	Every chart on the page follows it
The **Time range selector** component	`range`	Buttons from **Offered ranges**; `15m`, `1h`, `8h`, `24h` when empty
The **Tabs** component	The parameter in **Page parameter set** (default `view`)	See The tabs component
The **Asset selector** / **Area selector** components	`asset` / `area`	One button per entry of **Choices**, each a path under **Root asset path**
The **Severity filter** / **Asset filter** components	`filterSeverity` / `filterAsset`	Narrow the page's alarm and event components; **All** clears the parameter

Parameters is one text box of `name=value` pairs separated by commas, such as `machine=Loader` or `asset={asset}, area=Line1`. Spaces around names and values are trimmed; a pair without `=` is ignored.

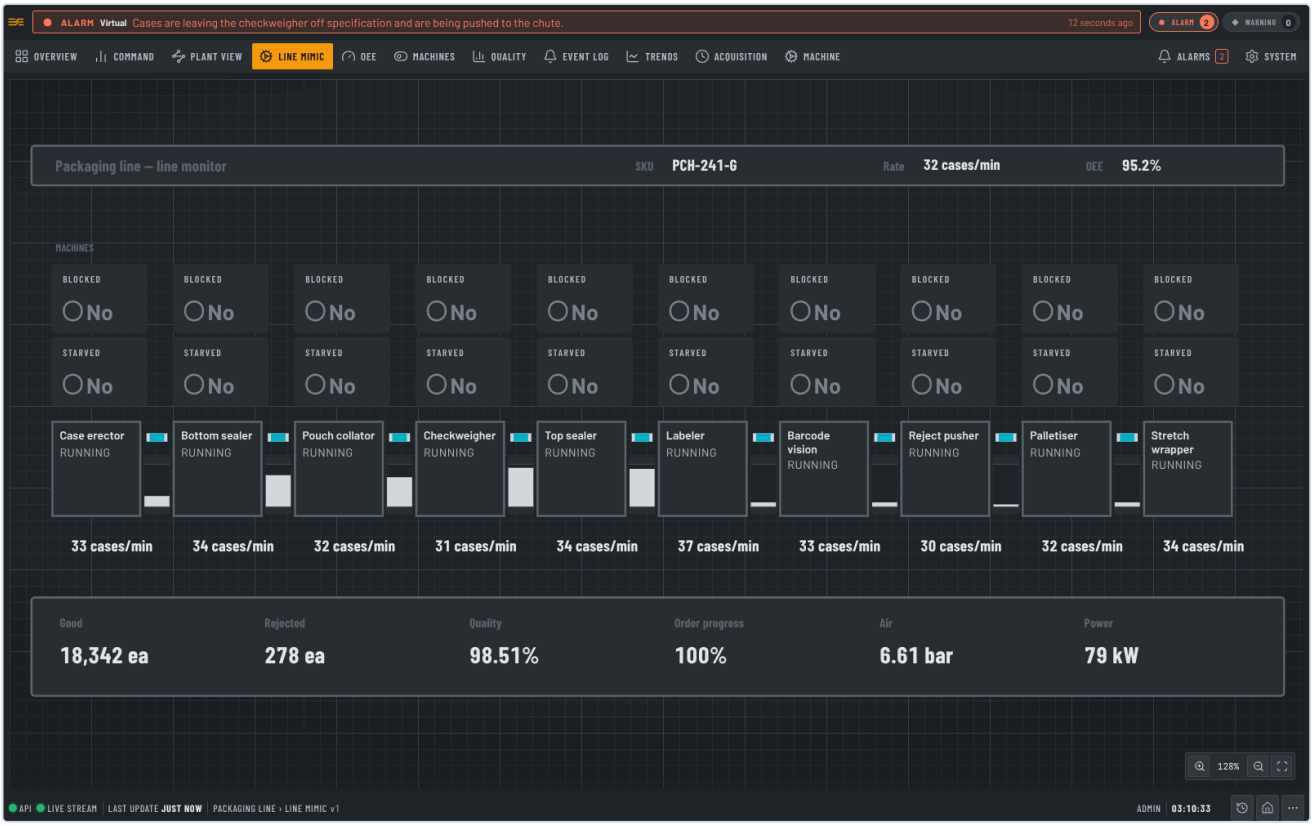
Inside an open card or dialog, an interaction that changes a parameter, or opens another page, changes the card rather than the page beneath it. Each such step can be undone with the card's back arrow or with `Esc` (see Keys in the operator screens).

Open a page with **Parameters** shows the target page with the parameters already in effect, the listed ones on top, and puts the listed ones in the address, as in `/page/machine?machine=Loader`. The address is read back: opening that address from a bookmark, or going back to it with the browser's back arrow, shows the **Machine** page for the loader again. So the **Line mimic**'s machine blocks could open the **Machine** page in full instead of as a card, with the same **Parameters**.

Worked example: one Machine page for ten machines

On the demonstration's **Line mimic** page, each machine block (Case erector, Bottom sealer, … Stretch wrapper) is a process-machine component whose **Navigation** is:

Field	Value on the Loader block
Action	**Show a page as a card**
Target page	`machine`
Parameters	`machine=Loader`



The Line mimic page: ten machine blocks, each with its blocked and starved flags, state and rate; pressing a block opens the Machine page as a card for that machine

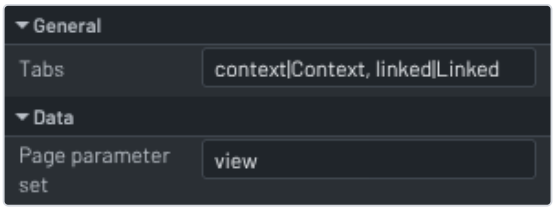
The **Machine** page declares `machine` (default `Loader`) and binds all its components through it: the state indicator to `Virtual/Computed/{machine}State`, the **Actual target** to `Virtual/Computed/{machine}Throughput` and `Packaging/{machine}/RatedSpeed`, the mini trend to the throughput, and the boolean track to `Packaging/{machine}/Faulted`, `Starved`, `Running` and `Blocked`.

When an operator presses the **Checkweigher** block, a card opens with `machine` set to `Checkweigher`, and every binding on it reads the checkweigher's signals. Close it, and pressing the **Labeler** block opens the same page as a card for the labeler. The page was drawn once.

The tabs component

Tabs is a row of tab buttons that sets a page parameter. It does not contain other components. On a grid page a component belongs to the page, not to a container, so a tab works by switching a parameter, and each component you want on a tab carries a visibility rule that reads that parameter.

Property	Meaning
Tabs	The tabs, separated by commas. Write `value|Label` to make the parameter value differ from the word shown; a plain `value` is both
Page parameter set	The parameter the tabs write. Default `view`



The settings of the Tabs component on the reference works' Batch page: Tabs context|Context, linked|Linked in the type's General group, and Page parameter set view under Data

The tab whose value matches the parameter's current value (ignoring case) is drawn as selected. With no **Tabs**, the component says *No tabs configured*.

Worked example: Context and Linked views

The reference demonstration (`examples/demo/reference.meelmohmi` , page `reference-batch`) does this:

1. The page declares a parameter `view` with default `context` , so the first tab is selected when the page opens.
2. A **Tabs** component has **Tabs** `context|Context, linked|Linked` and **Page parameter set** `view` .
3. The context summary has the visibility rule **Page parameter equals**, **Parameter** `view` , **Value** `context` .
4. The relationship list has **Page parameter equals**, **Parameter** `view` , **Value** `linked` .

Pressing **Linked** sets `view` to `linked` : the context summary disappears and the relationship list appears.

On a page that has no declared default (one you created in the Studio), nothing is selected until somebody presses a tab, and a component shown only when `view` equals `context` stays hidden. With two tabs, give the first tab's components the rule **Page parameter equals** `view` = `linked` and turn on **Hide when this is true, instead of showing**. They are then shown until **Linked** is pressed.

Conditions and colours in one place

Four mechanisms decide what a component looks like. They are independent, and each has its own editor:

Mechanism	Decides	Reads	Evaluated
Visibility	Whether the component is drawn at all	Roles, a signal, an asset's execution or state, alarm count, a page parameter, the screen size	One rule per component; the switch inverts it. A **Critical** component is always drawn
Thresholds	The colour, label, icon and emphasis of a value	The component's own value	Bands in order; the **first** match wins
Value state	Whether the value is shown as stale, uncertain, bad or missing	The value's quality and age	Before thresholds: a value that is not good and fresh takes its state colour and no band
Animation bands (shapes and models)	Fill, stroke, blink or state word of a shape or part	The animation's source	Each animation in order; within one animation the first band wins; a later animation of the same kind overrides an earlier one

Practical consequences:

- A stale or bad reading never lights a threshold band, never satisfies **Signal compares**, and drives no geometry. It may still colour a shape's fill or stroke, in the state colour.
- Put the most severe threshold band first.
- A rule that compares a signal (**Signal compares**, **Signal is true**) is false while the signal is not live, so a panel shown only "while the pump is running" hides when the pump signal is lost.

Moving a project: export and import

What a .meelmohmi file contains

Export in the project browser writes the selected project as `{project id}.meelmohmi`, a JSON document with:

Part	Contents
Schema version	The version of the page format that wrote it
Project	Name, settings, theme, navigation, strings
Pages	The **published** version of each page. Drafts are not exported
Composites	The reusable components the project uses
Templates	The templates this site added. The built-in ones are left out, because the receiving installation has its own
Assets	The **identity** of every asset a component's media field names: id, file name, type, size, uploader. **Not the bytes**

Export needs **Edit pages**.

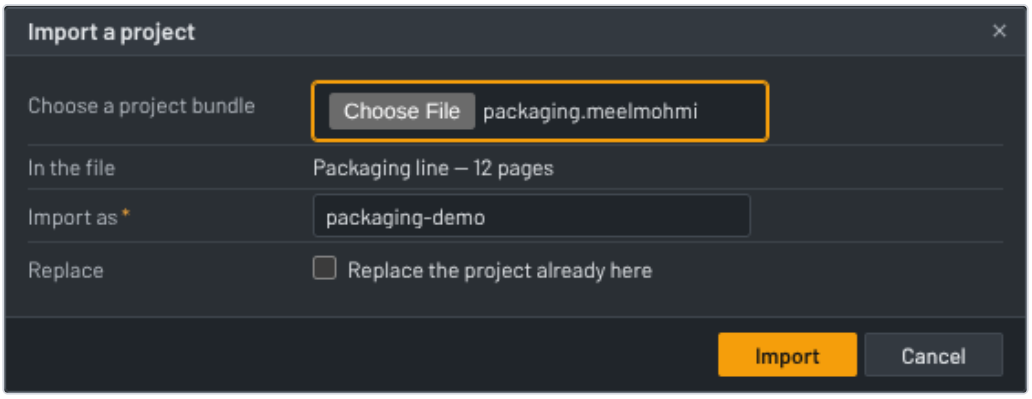
The asset bytes stay out so that a project with one machine model is not a 30 MB text file. Because an asset's id is the hash of its bytes, the receiving installation can still say exactly which files it lacks, and uploading the same file there repairs the reference.

The asset list covers the media fields of components (**Image asset**, **Image**, **Model**). A texture in **Part appearance** and a page's **Background picture** are not in it and are not reported as missing, so copy those images across yourself.

Importing

Open the project browser and press **Import**. The **Import a project** dialog:

Field	Meaning
Choose a project bundle	A `.meelmohmi` (or `.json`) file
In the file	What the file holds, for example *Packaging line — 12 pages*, read before anything is written
Import as	The project id to create. Defaults to the file's; change it to import one line several times
Replace	**Replace the project already here**. Without it, an existing id is refused with *A project called 'packaging-demo' already exists. Import it under a different id, or say to replace it.*



The Import a project dialog with packaging.meelmohmi chosen: In the file reads Packaging line — 12 pages, Import as packaging-demo, and Replace the project already here not ticked

Import needs **Create pages**, **Edit pages** and **Publish pages**.

The import goes through the same path as authoring: composites first, then templates, then the project, then each page is saved as a draft, given any breakpoint layouts it lacks, validated, and published with the note *Imported from* '*packaging-demo*'. A page that fails validation is not refused: it stays as a draft so you can open it and fix it.

When it finishes the dialog shows the report:

Line	Meaning
Published: *n*	Pages published
Left as drafts: *n*	Pages that did not validate, each with its errors. A signal or asset that existed at the source and not here is the usual cause
Missing media: *n*	Assets the pages name that this installation does not have, by file name where the bundle says it
A warning about a newer schema	The file was written by a newer version; anything this version does not know was dropped

To repair **Missing media**, upload each listed file through any media field (step 3 of the 3D worked example). The same file gets the same id, so the imported pages find it without further edits. Then publish the pages that were left as drafts.

Keys in the Studio

Shortcuts use `Ctrl`; on a Mac, `Cmd` does the same. The canvas shortcuts work while the canvas has focus (click an empty part of it, or `Tab` to it) and do nothing on a page you are viewing read-only.

Everywhere in the Studio

Keys	Does
`Ctrl` + `K`	Opens or closes the command palette
`Alt` + `1` … `Alt` + `9`	Goes to the first to ninth destination of the top bar, counting the work areas and then the administration screens. Ignored while typing in a field

The canvas

Keys	Does
`V`	**Select** tool
`R`	**Rectangle**
`E`	**Ellipse**
`L`	**Line**
`P`	**Run** (a line with corners)
`N`	**Polygon**
`K`	**Arc**
`T`	**Text**
`S`	**Symbol**
`I`	**Icon**
`M`	**Image**
`Esc`	Cancels a drag in progress (the component goes back where it was), puts the tool down and clears the selection
`Delete` or `Backspace`	Deletes the selection
Arrow keys	Move the selection: one grid cell on a grid page; one **Snap spacing, 0 for none** step on a freeform page (8 px when it is 0)
`Shift` +arrow keys	Grid page: make the selected component one cell wider, narrower, taller or shorter
`Ctrl` + `←` / `Ctrl` + `→`	Turn the selection 1° anticlockwise / clockwise
`Ctrl` + `Shift` + `←` / `→`	Turn the selection 15°
`Ctrl` + `Z`	Undo
`Ctrl` + `Shift` + `Z` or `Ctrl` + `Y`	Redo
`Ctrl` + `C`	Copy the selection (inside the Studio, not to the system clipboard)
`Ctrl` + `X`	Cut
`Ctrl` + `V`	Paste, with new ids, offset so the copy is visible
`Ctrl` + `D`	Duplicate
`Ctrl` + `G`	Group the selection
`Ctrl` + `Shift` + `G`	Ungroup
`Ctrl` + `A`	Select every component on the page
`Shift` + `O` / `Shift` + `P`	Select the previous / next component in draw order, to reach a shape hidden under another

The tool letters work only on freeform pages and only without `Ctrl`, `Alt` or `Shift`. `Tab` is not taken by the canvas: it moves focus out, as everywhere else.

Pointer with keys

Gesture	Does
`Shift` +click or `Ctrl` +click	Adds the component to the selection, or removes it
`Shift` +press on a component	Adds it to the selection without starting a drag
`Alt` +click on a grouped component	Selects that member alone instead of its whole group
`Shift` +drag a marquee (or `Ctrl`)	Adds what it encloses to the selection
`Shift` while drawing	A line or run snaps to 45° steps; other shapes are drawn square
`Shift` while dragging a resize handle	Keeps the proportions (freeform)
`Shift` while dragging the rotate handle	Turns in 15° steps
Double-click the rotate handle	Sets the rotation back to 0°
Double-click a vertex	Removes it
`Ctrl` +wheel	Zooms the freeform canvas about the pointer, within the page's zoom limits
Middle button drag	Pans the freeform canvas

Lists, pickers, menus and dialogs

These keys work the same throughout the browser client, including the operator screens.

Where	Keys	Does
Trees (project browser, **Signals**, **Layers**, **Navigation**)	`↑` `↓`, `Home` `End`	Move; with `Shift`, extend the selection
	`Page Up` `Page Down`	Move ten rows
	`→` / `←`	Open / close a branch, or go to the first child / the parent
	`Enter`	Activate the row (open it), or open/close a branch
	`Space`	Select or tick the row
	`*`	Open every sibling branch
	`Ctrl` + `A`	Select all, where several may be selected
	Letters	Jump to the next row starting with what you type
	`Menu` or `Shift` + `F10`	Open the row's menu
Pickers (signals, assets, media)	`↑` `↓`, `Home` `End`	Move the highlight
	`Enter`	Take the highlighted match (never the typed text)
	`Esc`	Close the list

Where	Keys	Does
Menus	`↑` `↓`, `Home` / `Page Up`, `End` / `Page Down`	Move
	`Enter` or `Space`	Choose
	`→` / `←`	Open / close a submenu
	`Esc`	Close (a submenu first) and return focus
	`Tab`	Close every menu
	Letters	Jump to the next item starting with what you type
Dialogs	`Enter` in a field	Presses the confirm button (not in a text area)
	`Esc`	Cancels
	`Tab` / `Shift` + `Tab`	Moves through the dialog's controls, wrapping round
Toolbars (such as the editor toolbar)	`←` `→`, `Home` `End`	Move focus along the toolbar

Keys in the operator screens

Keys	Where	Does
`Ctrl` + `K`	Anywhere	Opens or closes the command palette. The operator screens have no `Alt` +digit jumps
`Tab` / `Shift` + `Tab`	Anywhere	Moves through the header, the page rail, the page's controls, the **Tabs** buttons and the status bar
`Enter` or `Space`	A focused tile, shape or process component that has an action	Does what pressing it does: opens a card, a page, a detail, or asks to write
`Enter` or `Space`	The alarm banner	Opens the alarm drawer
`Esc`	The alarm drawer	Closes it
`Esc`	The detail overlay (chart, legend, catalogue facts)	Goes back one step if it can, otherwise closes
`Esc`	A card or dialog	Goes back one view (after a drill-down or a parameter change inside it), otherwise closes it
`Menu` or `Shift` + `F10`	A row of the alarm drawer	Opens the row's menu: acknowledge, shelve, details
`Tab`, then `Enter`	A **Machine model (3D)** with **Part detail**	Each part with detail is also a button under the model, named after the part (*Open this part's detail*), so you can reach it without a pointer

Commands from the operator screens remain subject to the confirmation and the four gates described in the *The browser operator HMI* chapter; no key bypasses them.

Data access and SDKs

> **Module: Data access and SDKs** (`data-access-sdk`). This chapter covers the data-access server, its REST API under `/api/v1` , the service accounts integrations authenticate with, and the Python and .NET SDKs. It is present only on installations licensed for the Data access and SDKs module. On the **Modules** screen the module is listed as **Data access and SDKs** `data-access-sdk` with the capabilities `DataAccessServer` , `HistoricalApi` , `RealtimeApi` , `PythonSdk` and `DotNetSdk` .

Meelmo records the plant; other systems want to read it — a reporting tool, a data-science notebook, a manufacturing execution system, a Grafana dashboard, a nightly export to the data lake. The data-access module is how they do it: one API, one kind of credential, one set of limits, whichever client is asking.

Everything in this chapter is a read path. There is no write to a controller through the API, the SDKs, gRPC or Grafana. The only writes the API accepts are to Meelmo's own records — saving a dataset or report definition, adding an annotation — and each needs its own permission.

The examples use the demonstration packaging line: signals such as `Packaging/Checkweigher/CaseWeight` , `Packaging/Checkweigher/Rate` and `Packaging/Erector/Rate` , the saved datasets `packaging-line-shift` (every charted signal at one-minute cadence, good quality only) and `packaging-line-runs` (one row per order), and the reports `packaging-line-shift-report` and `packaging-line-run-report` . The server address used is `https://historian.plant.example:7443` .

The data-access server

The API is served by **the process that holds the node**:

Installation	Process	Default address
Plant server	the recorder service	`https://<server>:7443`
Workstation	the web host beside the node	`https://<workstation>:7443`
Beside a node whose history is in a database store (SQL Server, TimescaleDB, Oracle, QuestDB, ClickHouse)	the separate data-access server, registered by hand	`https://<server>:7445`

Setup turns the data API on (`"dataAccess": { "enabled": true }` in the recorder's configuration) wherever a browser host or the data-access component is installed, because the browser itself reads history through it. That process is the only one that can serve an embedded store's history *and* live values.

The separate data-access server (`Meelmo.DataAccess.Server`) has one use: several readers of a database historian. It serves history on port 7445, does not acquire, and so refuses live subscriptions with `no_live_feed` . It is not installed as a service by any installer; on Linux register `deploy/systemd/meelmo-dataaccess.service` , on Windows run `scripts/service/install-dataaccess-service.ps1` .

Properties every data-access server shares:

- **TLS or nothing.** The server refuses to start on a routable address without TLS.
- **Every response carries `schemaVersion` .**
- **Its own description.** Each host serves its OpenAPI description, unauthenticated, at `/openapi/v1.json` , generated from the routes, so it cannot describe an endpoint that does not exist.

- **gRPC** offers the same catalogue, history, real-time and status services, with the same tokens and limits, on a second port. The `.proto` ships with the release; field numbers are never reused.
- **Licensing.** A capability the licence lacks is refused with `not_licensed` (HTTP 402). This is not a payment prompt; retrying will not help.

The **Modules** and **License** screens show whether the module is on. After a new licence is installed, restart the host: the data-access server is composed when the host starts.

Service accounts

An integration authenticates as a **service account**, never as a person. Give every integration its own account: then one can be withdrawn without touching the others, and the node's audit trail says which integration did what. There is no shared key.

What an account holds

Part	Meaning
Name, owner, description	who it is and who to ask about it
Permissions	what it may do (table below)
Allowed signal prefixes	what it may read; empty means every signal
Denied signal prefixes	what it may never read, even under an allowed prefix. Deny beats allow
Limits	concurrent streams, signals per stream, signals per query, longest query in days
Expiry	the last day its token works, or none
Token	`<accountId>.<secret>`, for example `svc-943910060143.…`. Shown once

Permission	Browser label	Grants
`ReadCatalog`	**Read the catalog**	listing signals, history availability, the asset tree and types, dataset and report definitions, historian and licence status
`ReadHistory`	**Read history**	history queries, statistics, value-at, running a dataset
`ReadRealtime`	**Read live values**	current values, live subscriptions and the alarms currently raised
`ReadEvents`	**Read events**	runs, events, states, their context and links
`Export`	**Export**	dataset export to files, and rendering reports
`ManageDatasets`	**Manage datasets**	saving dataset definitions, their drops, question synonyms, process limits and cross-checks — where the node keeps them too, as this account (and, for older integrations, report definitions)
`WriteAnnotations`	**Write annotations**	adding to the record of what happened
`Administer`	**Administer**	the administration routes, including minting and revoking other accounts. The one grant that can widen every other
`Mcp`	—	being used by an AI assistant over MCP; part of the Industrial integration module, ticked in the Desktop's window and in no preset

`Reader` is the bundle of the four read permissions, and it is what most integrations need.

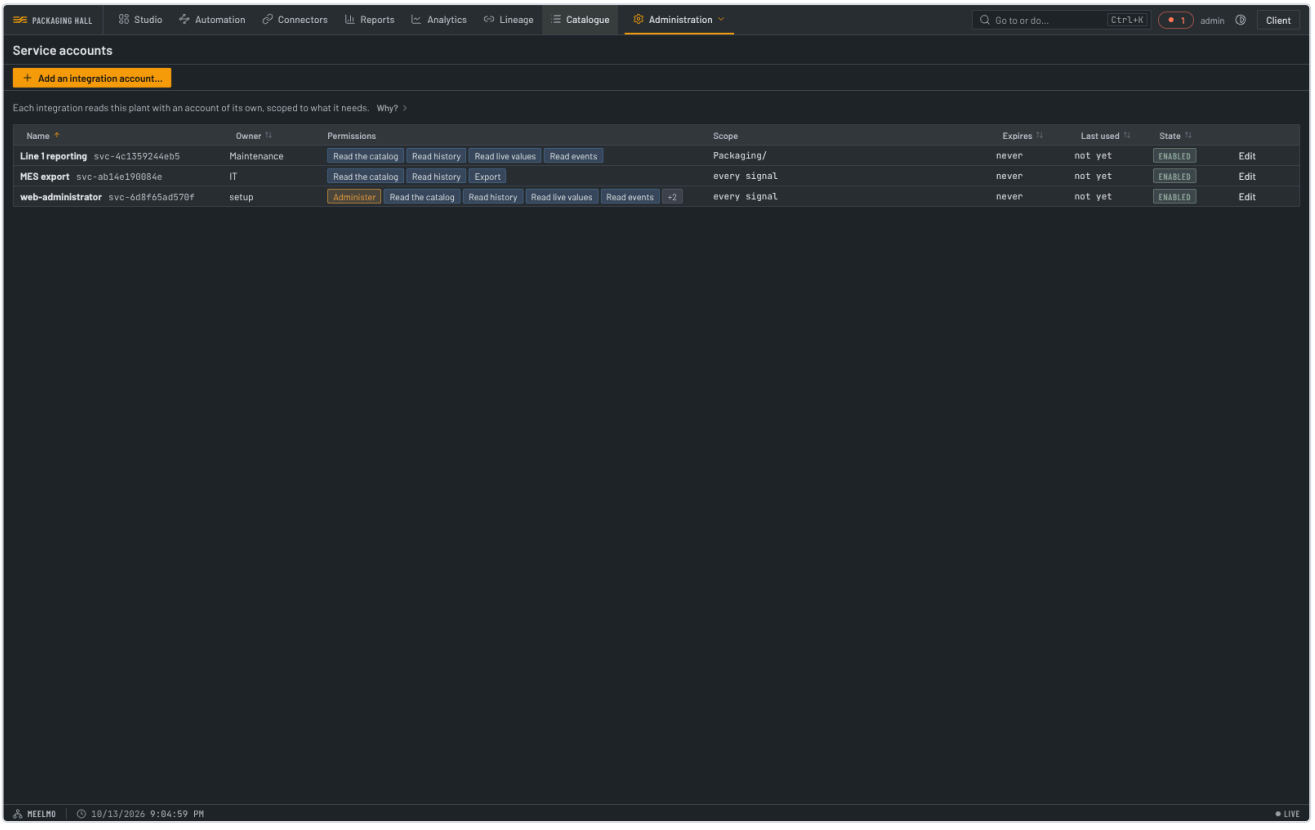
Scope narrows rather than fails. An account allowed only `Packaging/Checkweigher` that asks for the whole plant gets the checkweigher's signals, not an error it will retry for ever. Check what came back rather than assuming everything you named did. The same scope covers the plant model: an asset, run, event, state or alarm is shown only when the account may read every signal under the asset it belongs to.

A readable-from date bounds every read, including calendar buckets that start before it; a gap in the series says which part was not read.

The token is shown once. The node keeps only a salted digest (a 256-bit secret, stored as salted SHA-256), so there is nothing to show again. A lost token is replaced by issuing a new one, which stops the old one at once. Revocation takes effect on the next request.

Creating an account in the browser

With the Web HMI module, **Administration ▸ Service accounts** in the browser creates, edits, rotates, disables and deletes accounts. The *Administration in the browser* chapter of that module describes the screen in full.

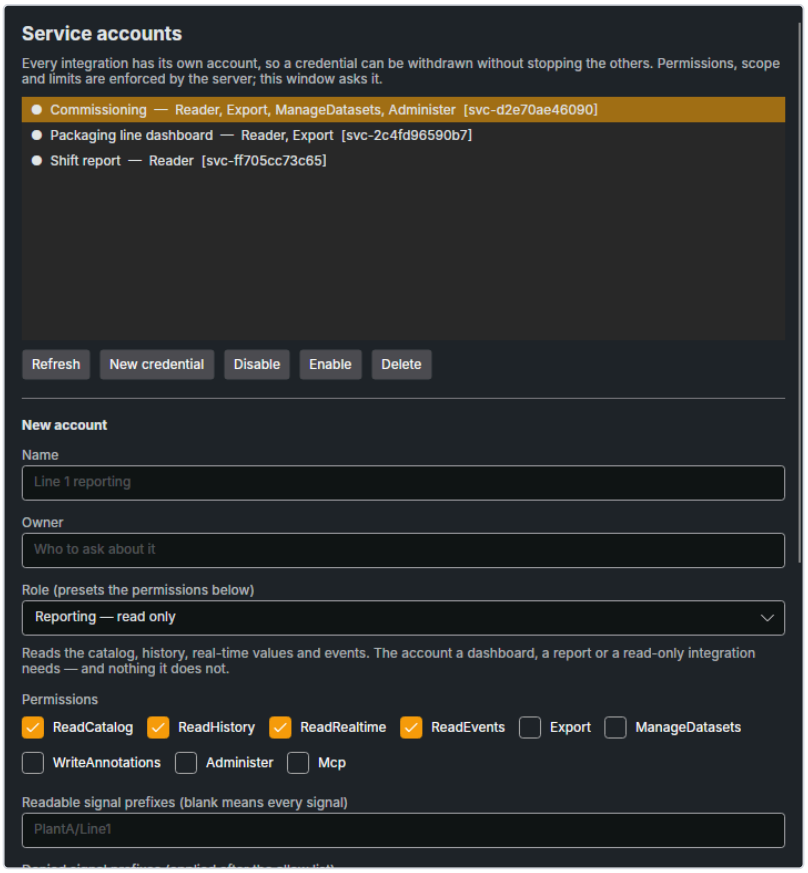


The browser's Service accounts screen: the web-administrator account setup created, its owner, its permissions led by Administer, its scope every signal, no expiry, not yet used, and enabled.

In short: **Add an integration account…**, fill in **Name**, **Owner**, **What it is for** and **Last day the token works**, tick the **Permissions**, mark the **Scope** on the catalogue tree (**It may read this**, **It may never read this**), set the **Limits**, and **Create the account**. The token appears in **The token for …**; **Copy the token**, then **I have it — hide it**.

Creating an account on the Desktop

On the Desktop, **Administration ▸ Service Accounts…** opens the same accounts. "Every integration has its own account, so a credential can be withdrawn without stopping the others. Permissions, scope and limits are enforced by the server; this window asks it."



The Desktop's Service accounts window: three accounts listed with their roles and ids, the Refresh, New credential, Disable, Enable and Delete buttons, and the New account form with the Reporting — read only role ticking the four read permissions.

The list shows each account as *name — permissions [id]*, for example `Packaging line dashboard — Reader, Export [svc-943910060143]`. Under it:

Button	Does
Refresh	reads the accounts again
New credential	issues a new token for the selected account; the old one stops working
Disable / **Enable**	refuses or restores the account's token
Delete	deletes the account after a warning; anything using it stops at once

New account:

Field	Meaning
Name	for example `Line 1 reporting`
Owner	who to ask about it
Role (presets the permissions below)	**Reporting — read only** (the four read permissions), **Reporting with export** (plus Export), **Annotator** (catalogue, history and annotations), **Dataset author** (catalogue, history, events, Manage datasets), **Administrator** (Administer), **Custom — choose below** (ticks nothing). A role is a starting point: the boxes can be changed afterwards
Permissions	`ReadCatalog`, `ReadHistory`, `ReadRealtime`, `ReadEvents`, `Export`, `ManageDatasets`, `WriteAnnotations`, `Administer` (and `Mcp` where the Industrial integration module is present)

Field	Meaning
Readable signal prefixes (blank means every signal)	for example `Packaging/Checkweigher`
Denied signal prefixes (applied after the allow list)	for example `Packaging/Checkweigher/WeightTarget`
Credential expires after (days)	blank for no expiry
Create account	creates it and shows the token

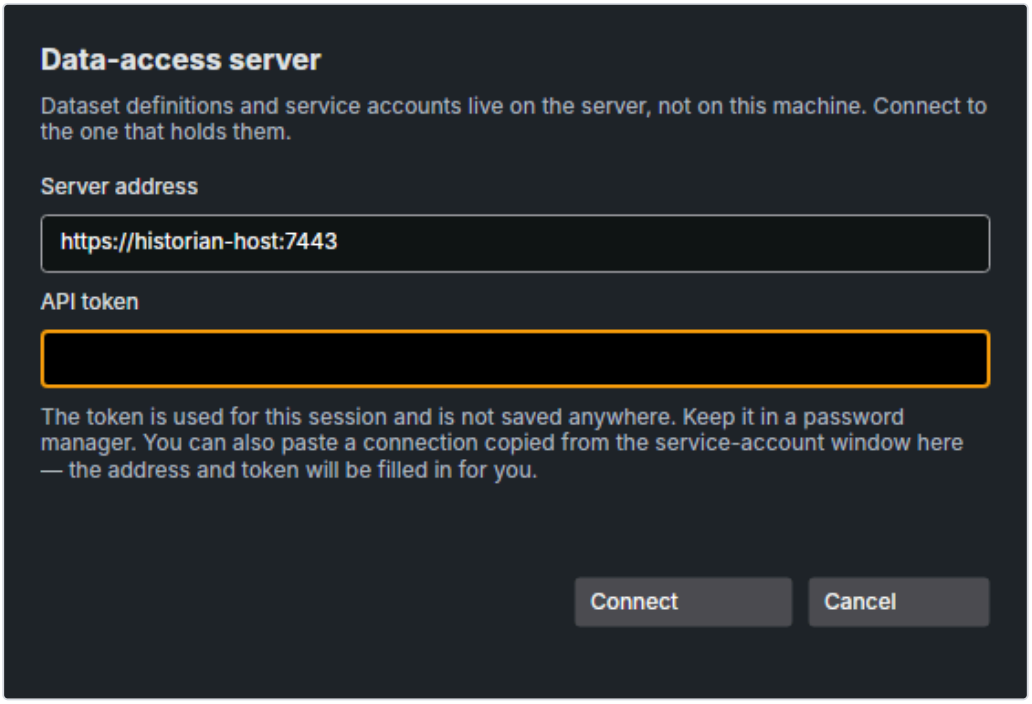
When the token is shown: "Copy this now. It is shown once and cannot be retrieved again — the server keeps only a hash of it." Two buttons copy it:

- **Copy token** — the bare token, for an integration's configuration.
- **Copy connection (address + token)** — the server's address and the token together, as one piece of text: `{"opsconn":1,"url":"https://historian.plant.example:7443","token":"svc-943910060143.…"}`. Pasting that into a Desktop's connect dialog fills in both fields at once, so a token never ends up next to the wrong address.

I have copied it closes the display.

Connecting a Desktop to the data-access server

Dataset definitions, report definitions and service accounts live on the server, not on the engineering workstation. When a Desktop window that reads them needs the server and this session holds no credential, the Desktop asks:



The Desktop's Data-access server dialog: the server address, an empty API token field, the note that the token is used for this session only and that a connection copied from the service-account window can be pasted here, and Connect and Cancel.

Field	Meaning
Server address	for example `https://historian-host:7443`
API token	the service account's token. It is used for this session and not saved anywhere — keep it in a password manager. A connection copied with **Copy connection (address + token)** can be pasted here instead; address and token are filled in
Connect	makes one authenticated call (listing the catalogue) so a wrong token fails here, not in the next window. A refusal shows the server's own code and message, for example `unauthenticated: …`. "Enter both the server address and a token." if either is missing; "The server could not be reached." if the address does not answer

Worked example: an account for the quality team's notebook

1. Browser: **Administration ▸ Service accounts ▸ Add an integration account…** (or Desktop: **Administration ▸ Service Accounts…**, **New account**).
2. **Name** `Quality notebook`, **Owner** `Quality engineering`, **What it is for** `Checkweigher analysis in Python`.
3. Permissions: **Read the catalog**, **Read history**, **Read events** and **Export**; untick **Read live values** — the notebook reads the past.
4. Scope: allow `Packaging/Checkweigher` and `Packaging/Line`; deny `Packaging/Checkweigher/WeightTarget`.
5. Limits: **Longest query, in days** `90`; **Last day the token works** the end of the quarter.
6. Create it, copy the token and store it in the team's secret manager as `MEELMO_TOKEN`.
7. The node's audit trail now shows `data_account.create` under your name.

The REST API

Everything lives under `/api/v1`. The token goes in the `Authorization` header as a bearer token:

```
export MEELMO=https://historian.plant.example:7443
export MEELMO_TOKEN='svc-1a2b3c4d5e6f.xxxxxxxxxxxxxxxxxxxxxxxx'

curl -sS "$MEELMO/api/v1/catalog/signals?search=Checkweigher" \
     -H "Authorization: Bearer $MEELMO_TOKEN"
```

To generate a client from the specification:

```
curl -sS "$MEELMO/openapi/v1.json" > meelmo-openapi.json
```

Routes

Area	Route	Permission	Does
Catalogue	`GET /catalog/signals` (`?search=`, `owner`, `term`, `limit`, `offset`)	ReadCatalog	lists signals; `search` also matches the glossary terms, synonyms and owners linked to a signal, `owner` and `term` narrow to one owner and one glossary term, whole
	`GET /catalog/glossary`	ReadCatalog	the glossary terms, and the owners and steward notes, of what the account may read
	`POST /history/availability`	ReadCatalog	per signal: first and last sample, approximate count, gaps, sources
History	`POST /history/query`	ReadHistory	reads history (below)
	`POST /history/statistics`	ReadHistory	count, min, max, average, standard deviation, RMS, first, last, good and bad counts
	`POST /history/evidence`	ReadHistory	whether a window is fit to quote from: readings against what the rate expects, quality counts, gaps, whether the read would be cut or bucketed
	`GET /history/value-at?signal=…&at=…`	ReadHistory	one value at one instant
Live	`POST /realtime/current`	ReadRealtime	each signal's latest value
	`GET /realtime/subscribe` (WebSocket)	ReadRealtime	live frames
Datasets	`GET /datasets/definitions`	ReadCatalog	saved dataset definitions

Area	Route	Permission	Does
	`POST /datasets/definitions`	ManageDatasets for a token; for a signed-in person where the node keeps the definitions, the node permissions `Config.Io` and `Config.Activate`	saves a definition
	`POST /datasets/query`	ReadHistory	runs a dataset and returns its rows as JSON
	`POST /datasets/export`	Export	runs a dataset and returns a file (`csv` , `arrow` , `parquet`)
Reports	`GET /reports/definitions` , `GET /reports/definitions/{name}`	`Reports.View` (or ReadCatalog)	lists and reads report definitions
	`POST /reports/definitions`	`Reports.Design` (or ManageDatasets)	saves one; changing its schedule or delivery also needs `Reports.Schedule`
	`DELETE /reports/definitions/{name}`	`Reports.Manage` (or ManageDatasets)	deletes one
	`POST /reports/render`	`Reports.Run` (or Export)	renders a saved report, or one sent inline, as `html` , `pdf` or `xlsx`
Semantic model	`GET /semantic/assets` , `GET /semantic/types`	ReadCatalog	the asset tree and the types
	`GET /semantic/context` , `GET /semantic/relationships`	ReadEvents	context values and links
	`POST /semantic/executions` , `/semantic/events` , `/semantic/states`	ReadEvents	runs, events and states in a window, by type, asset or attribute
Status	`GET /historian/status`	ReadCatalog	ingestion, storage and spool state
	`GET /license/status`	ReadCatalog	what this installation is permitted
	`GET /alarms/active`	ReadRealtime	what is currently raised

Area	Route	Permission	Does
	`GET /alarms/performance` (`from`, `to`, `area`)	ReadEvents	how the alarm system performed over a period, against ISA-18.2 and EEMUA 191: rates, floods, top contributors, chattering, fleeting and stale alarms, shelving, priorities, times to acknowledge and return, pairs that raise together; findings worst first (see *Alarms*, *Alarm performance*)
	`GET /diagnostics`	Administer	query latency, live subscriptions
	`GET /uns/broker`	Administer	the node's own MQTT broker: endpoints, counters, clients and their subscriptions, topics and retained values, refusals by reason. Read-only; `limit` cuts each list (default 200, at most 1 000)
Administration	`GET/POST /admin/service-accounts`, `POST /admin/service-accounts/{id}/rotate`, `POST /admin/service-accounts/{id}/enabled`, `DELETE /admin/service-accounts/{id}`	Administer	the service accounts themselves
Lineage	`GET /lineage/items` (`?search=`, `kind`, `limit`, `offset`)	ReadCatalog	the connections, tags, signals and everything that reads them, from the node's activated configuration
	`GET /lineage?item=…` (`direction=upstream|downstream|both`, `depth`, `limit`, `unread`)	ReadCatalog	what an item depends on and what depends on it, nearest first, with the edges between them; for a connection, `unread=true` also adds the device's tags no signal reads (see below)

Area	Route	Permission	Does
	`GET /lineage/impact?item=…` (`change=removed|disabled|renamed|retyped|moved`, `limit`, `offset`)	ReadCatalog	every dependant a change would reach, and whether it would read nothing afterwards
	`GET /lineage/graph`	ReadCatalog	the whole lineage the account may see, every node and edge — what a web host serving this node narrows for its own callers; a screen asks the routes above
Drift	`GET /drift`, `GET /drift/{connectionId}` (`limit`, `offset`)	ReadCatalog	each connection's tag table against its baseline, and one page of the differences
	`POST /drift/{connectionId}/check`, `POST /drift/{connectionId}/accept`	Administer	browses the device now; makes the latest browse the baseline (recorded as `api.drift.check` and `api.drift.accept`)
Glossary	`POST /project/glossary/terms`	ReadCatalog, and the node's permission to change configuration	writes one term (or, with `"remove": true`, takes it out) into the node's configuration draft and returns a review; activates nothing
	`POST /project/glossary/stewards`	the same	writes the owner and steward notes of one signal or asset into the draft; blank owner and notes take the entry out
	`POST /project/withdraw` (`kind: "glossary"`), `POST /project/drafts/{draftId}/activate`	the same	takes a staged glossary entry back out; activates the reviewed draft
Analysis	`POST /analysis/correlations`, `/analysis/occurrences`, `/analysis/contribution`, `/analysis/shapes`, `/analysis/run-comparison`, `/analysis/fingerprint`, `/analysis/line-time`, `/analysis/advisory-replay`	ReadHistory	the analyses the Analysis pages and the Desktop's Analysis Studio use

The lineage and drift routes work on every host that serves a node: the node's own data API, a web host or Desktop holding the node, and a web host attached to a node another process holds — a plant server, where the recorder service holds the node. Both are filtered like every route that names a signal: an account scoped to one line is not shown another line's signals, the tags and connections behind them, or anything reachable only through them. The .NET SDK reads them as `client.Lineage`.

On a web host whose node is elsewhere, the host reads the node's lineage graph (`GET /lineage/graph`) and its drift reports with its own machine account, then narrows them with the caller's account exactly as the node would, so a caller sees through the host what their account may see and nothing more. A device's unread tags are browsed by the node, after the host has checked that the caller may see the connection. **Check now** and **Accept as baseline** are not sent with the host's account: the host asks the node's control plane as the person signed in, so the node decides by their permission to change I/O and its audit trail names them (`drift.check`, `drift.accept`); a bearer token names no person and is refused there with `403 forbidden`. Only a server attached to no node at all answers these routes `503 not_served_here`, with a sentence saying so; `node_unavailable` means a node that did not answer.

The same holds for the changes only the node can make, asked of a web host attached to it: `POST /models/registry/{model}`, `POST /models/deployments/{signal}` and `…/rollback` are done at the node as the person signed in, with the node's permission to configure signals, the node's own reading of the file and the node's trail (`models.upload`, `models.deploy`, `models.rollback`); and `PUT /datasets/{id}/drops/{dropId}` asks the node, as the person, whether its `dataset-drop-roots.json` lets the drop write to the destination — a destination outside it is refused with `403` before anything is saved, as on the node, and a node that cannot say is answered `503 unavailable` rather than saved unchecked. A bearer token names no person and is refused there with `403 forbidden`.

On such a host these routes, and drift's **check** and **accept**, are gated on the person rather than on the account the session reads with: on a plant server every browser reads with one shared account that may only read. The host lets through a person whom the node granted the act's permission — `Config.Signals` for a model, `Config.Io` for drift, `Config.Edit` for a drop — as it re-reads that grant from the node on every request, and the node checks it again when it acts; `GET /dataset-drop-roots` is read at the node as the person, on the same grant as saving a drop. A node of a build from before these could be asked of it is answered `503 not_served_here`, saying that the node needs updating.

`GET /lineage?item=connection:{id}&unread=true` adds the tags the device offers that no configured tag reads, as leaves of kind `unreadTag` downstream of the connection. They come from a browse of the device: the drift monitor's latest browse when it is under 15 minutes old, otherwise a new one on a driver session of its own that waits at most 20 seconds, queued behind any other browse on the node so acquisition is never touched. At most 250 are added, in address order. The answer's `unread` object says what happened — `status` is `included`, `unsupported` (the driver cannot list its device's tags), `failed` (the device refused or did not answer in time, with `message`), `hidden` (the account may not read any of the connection's tags) or `unavailable` (the server is attached to no node) — with `total`, `shown`, `browsedUtc` and `reused`. A browse that reached the device is audited as `api.lineage.browse`; one answered from a recent browse is not, since it touched nothing.

The glossary, owners and steward notes

The glossary is part of the project's semantic model, beside the rules, the role catalogue and the asset model: terms with a definition and synonyms, linked to signals and assets, and for any signal or asset an owner (a person or a team, in the site's own words) and steward notes. It is held in the activated revision, so it drafts, diffs, activates, rolls back and is audited like the rest of the semantic section, and the Desktop's **Export semantic model** carries it (a file with a glossary is format 2; a build that predates it refuses such a file by name rather than dropping the vocabulary).

`GET /catalog/glossary` answers with every term and only the links and owners of what the account may read: an account scoped to one line sees a term's definition, not which signals on another line carry it. Every catalogue search — this route, gRPC, Grafana's variable query, the signal-health list and MCP's `catalog_search` — matches a signal's linked terms, synonyms and owner as well as its own fields, after the account's narrowing.

A glossary change goes into the node's configuration draft **as the person asking**, exactly like a model or a rule kept from an analysis page: the first request stages it and returns what activating the draft would change, the second activates. A machine token names nobody and is refused by the node; who may stage and activate is the node's permission to change configuration (the one that edits the semantic model). A person may link only signals and assets they may read; links they cannot see that a term already has are kept, and a term holding any is not theirs to remove. Each step is recorded on the hash-chained trail (`api.project.glossary.stage`, `api.project.glossary.withdraw`, `api.project.activate`). The .NET SDK reads them as `client.Catalog.GetGlossaryAsync` and `client.Project.KeepGlossaryTermAsync` / `KeepStewardshipAsync`.

Where the node keeps the saved definitions — on the node's own data API (the recorder's, a Desktop's own), a web host holding the node, and a web host attached to it — saving or removing a dataset, a drop, a question synonym, a process limit or a cross-check activates a revision of the node's configuration, and the node decides who may:

- **A token** acts as its data account. The account needs `ManageDatasets`, the node checks it again for itself, and its trail records the revision under the account (`svc:<id>`) with the account's name and owner (`definitions.apply`). A dataset may name only signals the account can read itself. A web host attached to the node sends the token on to the node: the save is the account's there too, never the host's own account's.
- **A signed-in person** (a browser session, a Desktop) acts as themselves: the node asks for `Config.Io` and `Config.Activate`, and names them.

A browser session with nobody signed in to the node is refused, with a sentence saying so. A dataset drop saved through a web host attached to the node is still a person's act: the host asks the node, as the person, whether the drop's destination is allowed, so a token is told to sign in or to save the drop at the node's own data API. A server that keeps definitions in its own files asks `ManageDatasets` as before.

The report routes accept either the person-level reporting capability (for a browser or Desktop session) or the account permission that stood for it before reporting had its own, so a token that saved definitions with `ManageDatasets` still does. Where the node keeps the reports, it judges a token's change from what it holds and what it is sent: adding or changing a report needs `Reports.Design`, removing one `Reports.Manage`, and changing when one runs or where it is delivered `Reports.Schedule` — or `ManageDatasets`, which stands for all three.

Reading history

```
curl -sS -X POST "$MEELMO/api/v1/history/query" \
     -H "Authorization: Bearer $MEELMO_TOKEN" -H "Content-Type: application/json" \
     -d '{
           "signals": ["Packaging/Checkweigher/CaseWeight", "Packaging/Checkweigher/Rate"],
           "start": "2026-09-29T06:00:00Z",
           "end":   "2026-09-29T07:00:00Z",
           "interval": "1m",
           "alignment": "previous",
           "quality": "good",
           "maxPoints": 4000,
           "maximumGap": "5m"
         }'
```

Field	Meaning
`signals`	signal ids (use the `id`, not the display name — the id survives a rename)
`start`, `end`	ISO-8601, required
`interval`	`auto` (default), `raw`, a stored tier (`100ms`, `1s`, `1m`) or a duration (`500ms`)
`alignment`	`none` (default), `previous`, `next`, `nearest`, `linear`
`quality`	`all` (default), `good`, `excludeBad`
`maxPoints`	points per signal; default 4,000, ceiling 100,000
`maximumGap`	how far alignment may carry a value before the row is missing instead
`includeProvenance`	adds each point's source
`aggregation`	the bucket's value: `full` (default — the mean, time-weighted for a signal that reports on change), `average`, `min`, `max`, `first`, `last`, `sum`, `count`, `rms`, `stddev`, `twa` (time-weighted average), `twstddev`
`interpolation`	for a time-weighted value: `stepped` (default) or `linear`
`continuationToken`	the previous page's token, for the next page

The answer carries `resolution` and `resolutionReason` (why that tier was chosen), `pointCount`, `durationMs`, `sources`, `continuationToken` and one entry per signal in `series`, each with `signalId`, `dataType`, `unit`, `resolution`, `gaps`, `truncated` and `points`. Five things to get right about points:

1. **`t` is a .NET tick count** — 100-nanosecond intervals since 0001-01-01 UTC. Unix seconds are `(t - 621355968000000000) / 10000000`. The SDKs convert for you.
2. **`v: null` means missing, never zero.** Treat it as a hole.
3. **`q` is the quality code; `0` is good.**
4. **`syn: true` marks a synthesized value** — one alignment produced, not the controller.
5. **Buckets are not readings.** When `resolution` is not `raw`, each point is a bucket with `n` (count), `mn` / `mx`, `f` / `l` and more. A series with `timeWeighting` (`stepped` or `linear`) was weighed by time: each bucket then also has `twa` (the time-weighted average), `twt` (ticks a value held), `twi` (the area, value·seconds) and `twd` (the spread over time), and a bucket with `n: 0` and `syn: true` is one no sample fell in that a value held through. `timeWeightingNote` says why a series could not be weighed. See *The historian*, *What a bucket's value is*.

Gaps are reported, not smoothed over: a `gaps` entry is a window no source could serve, or a window in which the node was not acquiring the signal (stopped, crashed, the connection disabled or restarted; see *The historian*, *When the node was not acquiring*). A gap may hold no readings at all or begin inside a bucket; draw no line across it. **Paging**: a series with `truncated: true` stopped at the point budget and the answer carries a `continuationToken`; send the same request again with that token to get the rest, repeating until no token comes back. The token is signed, bound to the request and good for an hour. An aligned read issues no token: page it unaligned and align the pages yourself.

Live values

`POST /realtime/current` with a list of ids returns each signal's value with **both** `sourceTimestamp` (when it was true at the controller) and `receiveTimestamp` (when the server saw it).

`GET /realtime/subscribe?signals=A,B&mode=changed` is a **WebSocket** endpoint. `mode` is `changed` (default), `sample`, `batch`, `batchinterval`, `deadband`, `interval` or `frame`; `batchinterval` is the lossless mode — every acquired sample, flushed on a fixed cadence. A frame may carry `droppedFrames`: the server shed

data because the client was not reading fast enough. Slow the subscription down with an interval or deadband, or record the hole. Only a host that acquires has live data; a history-only server answers `no_live_feed` (503).

Datasets

A **dataset** is a saved query that turns history into a table: which signals, at what cadence, aligned how, filtered to what quality, segmented by run, state or event. Datasets are defined on the Desktop (**Data ▸ Datasets**) or saved over the API. The base chapter *Datasets* describes them.

`POST /datasets/query` runs one and answers with its rows as JSON:

Field	Meaning
`definitionName`	a saved dataset's name...
`definition`	...or a definition sent inline
`start`, `end`	overrides the definition's calendar range

The answer carries `schema` (the columns, each saying which row array it is read from), `rowCount`, `truncated`, `warnings`, and `rows`, each with `t` (ticks) and the arrays the shape uses: `v` values, `q` qualities, `x` text, `c` context, `e` an end time and `s` a segment id for run- and state-shaped datasets. Rows are capped at 1,000,000 per query.

`POST /datasets/export` takes the same fields plus `format` (`csv`, `arrow` or `parquet`) and returns the file. It needs `Export`, and on the server writes only under its configured directory — a path in a request is never used as given.

Reports

`POST /reports/render` renders a report exactly as a schedule would:

Field	Meaning
`name`	a saved report...
`definition`	...or one sent inline, for a preview nobody saves
`start`, `end`	the window; both or neither. Neither takes the report's own look-back ending now
`format`	`html` (default), `pdf` or `xlsx`
`language`	which language titles are taken in
`parameters`	the report's parameters by name; a required one missing refuses the run

Errors

Every failure returns `{"code": "...", "message": "...", "detail": "..."}`. **Branch on `code`**; the message is for people and is reworded and translated.

Code	HTTP	Means	Do
`unauthenticated`	401	token absent, wrong, expired or revoked	get a new token
`forbidden`	403	authenticated but not permitted this operation or these signals	widen the account's permissions or scope

Code	HTTP	Means	Do
`not_licensed`	402	the licence lacks a capability	a person must change the licence; retrying will not help
`limit_reached`	429	over the per-minute budget, or too many failed tokens from this address; through a web host whose node is elsewhere, also the node's budget for that host	wait for `Retry-After`; if it follows 401s, fix the token first
`invalid_request`	400	malformed request	fix it; `detail` says what
`not_found`	404	no such signal, dataset or definition	
`no_live_feed`	503	this server holds history but does not acquire	connect to the acquiring host
`node_unavailable`	503	the node this host reads from did not answer	look at the node or the link to it, not this host
`not_served_here`	503	this host answered, and what was asked is done only where the node is held — a server attached to no node, a model change on a web host whose node is elsewhere, or a node too old to be asked by another host; the message says which	do it where the message says
`internal_error`	500	a fault in this host	collect the response and report it

Limits

Defaults; a licence may set them lower, and a service account's own limits lower still.

Limit	Default
Signals per query	5,000
Points per signal	100,000
Signals per subscription	2,000
Query range	1,100 days
Requests per minute per identity	600
Requests per minute for a web host's own account (`WebHost`)	30,000
Failed tokens per minute per address, before the next is refused unchecked	30
Rows per semantic query	10,000
Signals per catalogue request	5,000
Rows per dataset query	1,000,000
Rows per export	20,000,000
How long one query may run	5 minutes
Frames queued for one stream client before its policy applies	1,000
A stream with no client activity is closed after	10 minutes

The per-identity budget is kept per data account for a token and per browser session for a person signed in to a web host — the people a host serves read through one shared account, and one budget for all of them would have made each person's share depend on how many screens were open. A web host reads its node with one account for every browser it serves; setup mints that account with `WebHost`, which grants nothing to read or change and has the node budget it as a host rather than as one integration. The host shares among all its browsers what each asks of the node alike — the catalogue for ten seconds, the alarms and advisories for two. A refusal for a budget is `429 limit_reached` with `Retry-After` in seconds and a sentence a page shows in its reader's language.

What `/api/v1` promises

Within `v1`, across every release: a route keeps its path, method and meaning; response fields are added, never removed or retyped (your parser must ignore fields it does not know); request fields become optional, never mandatory; enumerations may gain members (treat an unknown member as unknown, not as an error); error codes and status codes are stable. If something must break, `/api/v2` appears beside `v1`, both are served for at least twelve months, and the deprecation is announced in the release notes, in `Deprecation` and `Sunset` headers and in the OpenAPI description.

Worked example: fetching a dataset with curl

The planner wants the early shift of `packaging-line-shift` — one row per minute of the charted signals, good quality only — first to look at, then as a Parquet file.

1. **List the datasets the account may see.**

   ```
   curl -sS "$MEELMO/api/v1/datasets/definitions" \
        -H "Authorization: Bearer $MEELMO_TOKEN" | jq '.[].name'
   ```

   The answer includes `"packaging-line-shift"`, `"packaging-line-runs"`, `"packaging-line-events"` and the other demonstration datasets.

2. **Run it over 06:00–14:00 and read the rows.**

   ```
   curl -sS -X POST "$MEELMO/api/v1/datasets/query" \
        -H "Authorization: Bearer $MEELMO_TOKEN" -H "Content-Type: application/json" \
        -d '{"definitionName":"packaging-line-shift",
             "start":"2026-09-29T06:00:00Z","end":"2026-09-29T14:00:00Z"}' \
        | jq '{rowCount, truncated, warnings, columns: [.schema[].name]}'
   ```

   `rowCount` is about `480` — eight hours at one row a minute. `columns` lists the charted signals and the transform the dataset adds (the first column's 15-minute rolling average). An empty cell is `null`: the dataset leaves a gap longer than five minutes empty rather than carrying the last reading forward.

3. **Export it as Parquet for the data lake.** The account needs **Export**.

   ```
   curl -sS -X POST "$MEELMO/api/v1/datasets/export" \
        -H "Authorization: Bearer $MEELMO_TOKEN" -H "Content-Type: application/json" \
        -d '{"definitionName":"packaging-line-shift",
             "start":"2026-09-29T06:00:00Z","end":"2026-09-29T14:00:00Z",
             "format":"parquet"}' \
        -o packaging-line-shift-2026-09-29-early.parquet
   ```

4. **Render the shift report as a PDF** over the last hour of that shift. The account needs **Export** (or the person, `Reports.Run`).

```
curl -sS -X POST "$MEELMO/api/v1/reports/render" \
     -H "Authorization: Bearer $MEELMO_TOKEN" -H "Content-Type: application/json" \
     -d '{"name":"packaging-line-shift-report",
          "start":"2026-09-29T13:00:00Z","end":"2026-09-29T14:00:00Z",
          "format":"pdf","language":"en"}' \
     -o shift-report.pdf
```

If a call answers `forbidden`, the account lacks the permission in the routes table; `not_found` means the dataset name is wrong or the account's scope hides it.

The Python SDK

```
pip install meelmo                      # the client itself: standard library only
pip install "meelmo[pandas]"            # to_pandas()
pip install "meelmo[polars]"            # to_polars()
pip install "meelmo[arrow]"             # to_arrow(), write_parquet(), query_arrow()
pip install "meelmo[realtime]"          # live subscriptions (websocket-client)
pip install "meelmo[all]"               # all of the above
```

From a checkout, `pip install ./src/Meelmo.DataAccess.Python`. Python 3.10 or later. Each conversion imports its library only when called.

```
import os
from meelmo import MeelmoClient

client = MeelmoClient("https://historian.plant.example:7443", token=os.environ["MEELMO_TOKEN"])
```

`timeout` (seconds, default 300) bounds each request. `verify_tls=False` exists for a development server with a self-signed certificate; it turns TLS into encryption without authentication, so do not use it anywhere else.

Call	Does
`client.catalog.signals()` / `.search_signals(term)`	lists the catalogue
`client.catalog.availability(ids)`	first and last sample, count, gaps
`client.history.query(signals, start, end, interval=, alignment=, quality=, max_points=, maximum_gap=)`	reads history; `result.to_pandas()` gives one column per signal
`client.history.query_paged(...)`	pages a long range by the server's continuation token; no point is yielded twice
`client.history.query_arrow(...)`	an Arrow table, skipping JSON — for large pulls
`client.history.statistics(signals, start, end, quality=)`	server-side statistics
`client.history.evidence(...)`	whether a window is fit to quote from
`client.history.value_at(signal, at, mode="before")`	one value at one instant
`client.datasets.definitions()`	saved datasets

Call	Does
`client.datasets.query(name=, start=, end=)`	a dataset's rows, as a `DatasetResult` (`columns` , `rows` , `row_count` , `truncated` , `warnings` , `to_pandas()`)
`client.datasets.export(path, name=, start=, end=, format=)`	a dataset written to a file
`client.realtime.current(signals)` / `.subscribe(signals, mode=, interval=)`	live values; gap frames are yielded, not swallowed
`client.admin.historian_status()`	samples received against persisted, dropped, spooled

Refusals raise `ApiError` with `status` , `code` , `message` and `detail` ; `PermissionError_` , `LicensingError` and `NoLiveFeedError` are subclasses for the ones you can act on.

Worked example: the shift dataset in pandas

```
import os
from datetime import datetime, timezone
from meelmo import MeelmoClient

client = MeelmoClient("https://historian.plant.example:7443", token=os.environ["MEELMO_TOKEN"])

start = datetime(2026, 9, 29, 6, 0, tzinfo=timezone.utc)
end   = datetime(2026, 9, 29, 14, 0, tzinfo=timezone.utc)

shift = client.datasets.query(name="packaging-line-shift", start=start, end=end)
if shift.truncated:
    print("Truncated:", shift.warnings)

df = shift.to_pandas()                  # indexed by timestamp, one column per dataset column
print(df.describe())

stats = client.history.statistics(       # raw history, good readings only
    ["Packaging/Checkweigher/CaseWeight"], start, end, quality="good")
for s in stats:
    print(s["signalId"], s["count"], s["average"], s["standardDeviation"])

client.datasets.export("shift.parquet", name="packaging-line-shift",   # the file for the data lake
                       start=start, end=end, format="parquet")
```

Read `truncated` and `warnings` before trusting the rows; a missing value is `None` (or `NaN` in pandas), never zero.

The .NET SDK

The `Meelmo.DataAccess.Client` package is a typed .NET client for the same API, read-only toward the plant. `new MeelmoClient(address, token)` exposes `Catalog` , `History` , `Datasets` , `Reports` , `RealTime` , `Semantic` , `Analysis` and `Admin` .

Member	Main calls
`Catalog`	`GetSignalsAsync()` , `SearchSignalsAsync(term)` , `GetAvailabilityAsync(ids)`

Member	Main calls
`History`	`QueryAsync(signals, start, end, interval, alignment, quality, maxPoints, maximumGap)`, `StatisticsAsync(...)`, `EvidenceAsync(...)`, `ValueAtAsync(signal, at, mode)`
`Datasets`	`ListDefinitionsAsync()`, `SaveDefinitionAsync(definition)`, `QueryAsync(DatasetRequest)`, `ExportToFileAsync(DatasetRequest, path)`
`Reports`	`ListDefinitionsAsync()`, `GetDefinitionAsync(name)`, `RenderToFileAsync(ReportRenderRequest, path)`, `RenderHtmlAsync(...)`
`RealTime`	`CurrentAsync(signals)` and a WebSocket subscription whose gap frames a caller should act on
`Semantic`	`GetAssetsAsync()`, `GetTypesAsync()`, `QueryExecutionsAsync(...)`, `QueryEventsAsync(...)`, `QueryStatesAsync(...)`
`Admin`	`HistorianStatusAsync()`, `LicenseStatusAsync()`, `ActiveAlarmsAsync()`, `ServiceAccountsAsync()` and the account operations

A refusal throws `MeelmoApiException`, whose `Error` carries the server's `Code` and `Message`.

Worked example: exporting the shift and the shift report

```
using Meelmo.DataAccess.Client;
using Meelmo.DataAccess.Contracts;

using var meelmo = new MeelmoClient("https://historian.plant.example:7443",
                                    Environment.GetEnvironmentVariable("MEELMO_TOKEN")!);

var start = new DateTimeOffset(2026, 9, 29, 6, 0, 0, TimeSpan.Zero);
var end   = start.AddHours(8);

// The checkweigher at one-minute resolution, good readings only.
var history = await meelmo.History.QueryAsync(
    ["Packaging/Checkweigher/CaseWeight"], start, end,
    interval: "1m", quality: "good");

foreach (var series in history.Series)
{
    Console.WriteLine($"{series.SignalId}: {series.Points.Count} points, {series.Gaps.Count}
gaps");
    foreach (var point in series.Points)
        if (point.V is { } value)                    // null is a hole, never zero
            Console.WriteLine($"{new DateTime(point.T, DateTimeKind.Utc):HH:mm} {value}");
}

// The saved dataset as a CSV file.
await meelmo.Datasets.ExportToFileAsync(new DatasetRequest
{
    DefinitionName = "packaging-line-shift",
    Start = start,
    End = end,
    Format = "csv",
}, "packaging-line-shift.csv");

// The shift report's last hour as a PDF.
await meelmo.Reports.RenderToFileAsync(new ReportRenderRequest
{
    Name = "packaging-line-shift-report",
    Start = end.AddHours(-1),
    End = end,
```

```
    Format = "pdf",
}, "shift-report.pdf");
```

Grafana

Grafana connects with no plugin through the **Infinity** data source: point it at the server, add the service account's token as a bearer header, and use

Route	For
`/api/v1/grafana/signals`	the signal list, for a dashboard variable
`/api/v1/grafana/query`	flat rows for a panel
`/api/v1/grafana/annotations`	executions, events, states and gaps as annotations

Add `&format=frame` to the query URL and set the Infinity query's format to **As Is**: the answer is then a Grafana data frame (`time` , `signal` , `value` , `text` , `quality` , `synthesized`) whose notices the panel shows in its header — series cut at the point budget, windows where nothing was recorded, signals the account may not read. Without it the same facts travel only in the `X-Meelmo-Truncated` , `X-Meelmo-Gaps` and `X-Meelmo-Denied-Signals` headers, where Grafana does not show them.

Give the dashboard its own account with **Reporting — read only** and a scope limited to the lines it shows.

Deploying integrations safely

- **One account per integration**, scoped to the prefixes it needs, with an expiry where the integration has an owner who can renew it.
- **Keep tokens in a secret manager.** The Desktop's connect dialog deliberately does not save them.
- **Never make a Meelmo process the internet-facing listener.** An integration outside the plant reaches the server through a reverse proxy; start the server with `MEELMO_API_TRUSTED_PROXIES` naming the proxy and have the proxy set `X-Forwarded-For` , or every client behind it shares one address and one client's failed tokens use up everybody's minute.
- **Direct database access bypasses Meelmo.** On a database historian, reporting views (`oa_v_signals` , `oa_v_samples` , `oa_v_rollup_1s` , `oa_v_rollup_1m` , `oa_v_events` , `oa_v_executions` , `oa_v_states`) can be created with the direct-database-access entitlement, but Meelmo cannot apply per-account signal scoping to somebody querying the database directly. Grant that as you would any other database access.
- **The MCP surface** (`POST /api/v1/mcp` , for AI assistants) belongs to the Industrial integration module, is off by default, and admits only accounts holding the `Mcp` permission.

MCP from a desktop assistant (standard input and output)

Claude Desktop, Claude Code, VS Code and Cursor start their MCP servers as local processes and talk to them over standard input and output. The data-access server's executable has a mode for exactly that:

```
Meelmo.DataAccess.Server mcp-stdio --node https://historian.plant.example:7443
```

It carries each JSON-RPC message, unchanged, to the node's `POST /api/v1/mcp` and writes the node's answer back. **It decides nothing itself**: the node authenticates the token, applies the account's scope, row limits and byte budget, offers only the tools that account may call — the same read-only tools as over

HTTP; there is no tool that writes — and records every answer on its trail, naming the assistant (`meelmo-mcp-stdio (claude-code)`) as the agent.

- **The token is never on the command line**, where every user of the machine can read it. Put it in `MEELMO_MCP_TOKEN` , or in a file named by `--token-file` or `MEELMO_MCP_TOKEN_FILE` . Without one the process does not start and says why on standard error; `--token` is refused.
- **The node's certificate is verified as the .NET SDK verifies it**, by the machine's trust store, with no way to switch it off. Install the site's root certificate on the machine. Plain `http://` is accepted only for a node on this machine's loopback.
- The account needs the **Mcp** permission and the node the Industrial integration module, as for MCP over HTTP. `--node` may instead be given as `MEELMO_MCP_NODE` .

It is installed with the **Server** component on Windows (`C:\Program Files\Meelmo\server\Meelmo.DataAccess.Server.exe`) and by the Linux package (`/opt/meelmo-server/Meelmo.DataAccess.Server`); in the container image it is `dotnet Meelmo.DataAccess.Server.dll mcp-stdio` .

Claude Desktop (`claude_desktop_config.json`):

```
{
  "mcpServers": {
    "meelmo": {
      "command": "C:\\Program Files\\Meelmo\\server\\Meelmo.DataAccess.Server.exe",
      "args": ["mcp-stdio", "--node", "https://historian.plant.example:7443"],
      "env": { "MEELMO_MCP_TOKEN_FILE": "C:\\Users\\me\\.meelmo\\assistant.token" }
    }
  }
}
```

Claude Code (a token file, so the token itself is written neither into the configuration nor into your shell's history):

```
claude mcp add meelmo --env MEELMO_MCP_TOKEN_FILE="$HOME/.meelmo/assistant.token" -- \
  /opt/meelmo-server/Meelmo.DataAccess.Server mcp-stdio --node
https://historian.plant.example:7443
```

or in a project's `.mcp.json` , which Claude Code asks you to approve once before it starts the server, and which expands `${HOME}` when it does:

```
{
  "mcpServers": {
    "meelmo": {
      "command": "/opt/meelmo-server/Meelmo.DataAccess.Server",
      "args": ["mcp-stdio", "--node", "https://historian.plant.example:7443"],
      "env": { "MEELMO_MCP_TOKEN_FILE": "${HOME}/.meelmo/assistant.token" }
    }
  }
}
```

VS Code (`.vscode/mcp.json` in the workspace; VS Code asks for the token once and keeps it in its own secret storage):

```
{
  "inputs": [
    { "type": "promptString", "id": "meelmo-token", "description": "Meelmo data-account token",
"password": true }
```

```
  ],
  "servers": {
    "meelmo": {
      "type": "stdio",
      "command": "C:\\Program Files\\Meelmo\\server\\Meelmo.DataAccess.Server.exe",
      "args": ["mcp-stdio", "--node", "https://historian.plant.example:7443"],
      "env": { "MEELMO_MCP_TOKEN": "${input:meelmo-token}" }
    }
  }
}
```

Cursor (`.cursor/mcp.json` in the project, or `~/.cursor/mcp.json` for every project). Cursor has no prompted inputs, so the token stays in a file:

```
{
  "mcpServers": {
    "meelmo": {
      "type": "stdio",
      "command": "C:\\Program Files\\Meelmo\\server\\Meelmo.DataAccess.Server.exe",
      "args": ["mcp-stdio", "--node", "https://historian.plant.example:7443"],
      "env": { "MEELMO_MCP_TOKEN_FILE": "${userHome}/.meelmo/assistant.token" }
    }
  }
}
```

In Claude Code, `claude mcp list` shows the server as connected once the token and the node are right, and `claude mcp get meelmo` shows how it is started. If the assistant shows the server as failed, its log holds the sentence the process wrote to standard error: no token, a token the node refused, a node that could not be reached, or a certificate the machine does not trust.

When it does not work

Symptom	Likely cause	What to do
`unauthenticated` on every call	token mistyped, expired, rotated or the account deleted	issue a new token; check **Expires** and **State** on the Service accounts screen
`unauthenticated` suddenly mid-session	somebody issued a new token or disabled the account	check the node's audit trail for `data_account.rotate` or `.disable`
`limit_reached` after a series of 401s	too many failed tokens from this address	fix the token, wait for `Retry-After`
fewer signals than asked for, no error	the account's scope narrowed the answer	widen **Scope** or use another account
`no_live_feed`	this server reads history but does not acquire	connect to the recorder or web host that holds the node
`node_unavailable`	the node behind this host did not answer	check the node and the link to it
`not_served_here`	this host is not where that is done	read the message: it names where
`not_licensed`	the licence lacks the capability	**License** screen; a person must change the licence

Symptom	Likely cause	What to do
TLS error from the client	the machine does not trust the site's root certificate	install the root certificate; do not switch verification off
an assistant lists the Meelmo server as failed	`mcp-stdio` found no token, or the node refused it	its standard error says which; set `MEELMO_MCP_TOKEN` or `--token-file`, and give the account the **Mcp** permission
the server will not start	a routable address without a certificate	give it a certificate, or bind to loopback
a series ends early	`truncated: true` at the point budget	follow the `continuationToken`, or use `query_paged`

SQL Server historian

> **Module: SQL Server historian (`historian-sqlserver`).** This chapter describes the SQL Server store for the historian. It is present only on installations licensed for that module. Everything about the historian that does not depend on the database — what is recorded, rollups, the history browser, the spool — is in the base chapter *The historian*; this chapter adds what is specific to keeping that history in Microsoft SQL Server.

What this module adds

The base product keeps history in the embedded store or in a SQLite file. This module adds a third place: a SQL Server database that your own database administrators run, back up and report against.

Choose it when:

- the site already runs SQL Server and wants plant history in the same estate, under the same backup, monitoring and security regime;
- reporting tools (Power BI, SQL Server Reporting Services, Excel, your own SQL) should read history directly through stable views rather than through the API;
- the history needs to be kept for years on storage the IT department manages.

Keep the embedded store when none of that applies. It needs no configuration and no database, and a node writing to it cannot lose its database.

What the SQL Server store does, in short:

Aspect	Behaviour
Ingestion	Bulk copy (`SqlBulkCopy`), one operation per typed sample table per batch, one transaction per batch. No statement is ever issued per sample.
Tables	Typed sample tables — numeric, integer, boolean and text — plus the signal catalogue, one-second and one-minute rollups, and the semantic model (events, executions, states, context, relationships).
Reads	Every query is parameterized. Signal ids are parameters; only the schema and table names the product owns are placed in SQL, each validated first.
Encryption	`Encrypt=True` is added to the connection string when you have not said otherwise.
Reporting	Seven stable `oa_v_*` views for your own SQL (see *Reporting views* below).
Late samples	Accepted. Rollups are refreshed with `MERGE`, so a bucket that later receives a late sample is corrected on the next maintenance pass.

Before you start

You need five things before the historian can write to SQL Server.

1. **The module, licensed and switched on.** The licence must grant the SQL Server provider capability, and the `historian-sqlserver` module must be enabled on the node under **Administration ▸ Modules**. A module arrives switched off; turning it on is a recorded configuration revision. Until both hold, **Microsoft SQL Server** is still listed in the store list, but marked *not available here* with the reason, and it cannot be chosen.

2. **The provider installed on the machine that runs the node.** The database stores are an optional installer component. If the Desktop says the provider is *not installed on this machine*, run the installer again and tick **Database history providers**. A node that is already writing to SQL Server is not affected by what is installed on a workstation.
3. **A supported server.** SQL Server 2019 or later.
4. **A database and a login.** Ask your DBA for an empty database (for example `MeelmoHistory`) and a login for the historian. The tables go into a schema of that database; the default schema name is `meelmo`.
5. **Network access from the node to SQL Server.** The node — not the Desktop and not the browser — opens the connection. Test it from the node's machine.

Permissions the login needs

Schema creation and running are two different jobs, and the product keeps them apart on purpose: a running historian that can create tables can also create them in the wrong database by accident.

Account	Used for	Grants
Administrator (once)	**Create schema** and the reporting views	`CREATE TABLE`, `ALTER` on the schema, and `CREATE VIEW` for the views
Runtime (the historian)	Writing samples, reading history, retention	`SELECT`, `INSERT` on the schema; `DELETE` for retention
Reader (optional)	Your reports and direct SQL	`SELECT` on the `oa_v_*` views only

The product ships these grants as a starting point (the logins are examples):

```
-- Runtime account (ingestion and queries). No DDL.
GRANT SELECT, INSERT ON SCHEMA::[meelmo] TO [meelmo_writer];
GRANT DELETE ON SCHEMA::[meelmo] TO [meelmo_writer];  -- retention only

-- Read-only account for reporting and direct SQL access.
GRANT SELECT ON SCHEMA::[meelmo] TO [meelmo_reader];

-- Schema creation, used once by an administrator and not by the service.
GRANT CREATE TABLE, ALTER ON SCHEMA::[meelmo] TO [meelmo_admin];
```

If your site does not want to give the historian's login any DDL rights at all, have the DBA enter the administrator credentials in the form, press **Create schema**, then put the runtime credentials back and save. Alternatively, grant `CREATE TABLE` and `ALTER` for the first start and narrow the login to read and write afterwards.

Validate (below) checks that the account has `SELECT` on the signal catalogue and `INSERT` and `DELETE` on the numeric sample table, by asking the server what it would allow rather than by trying and rolling back.

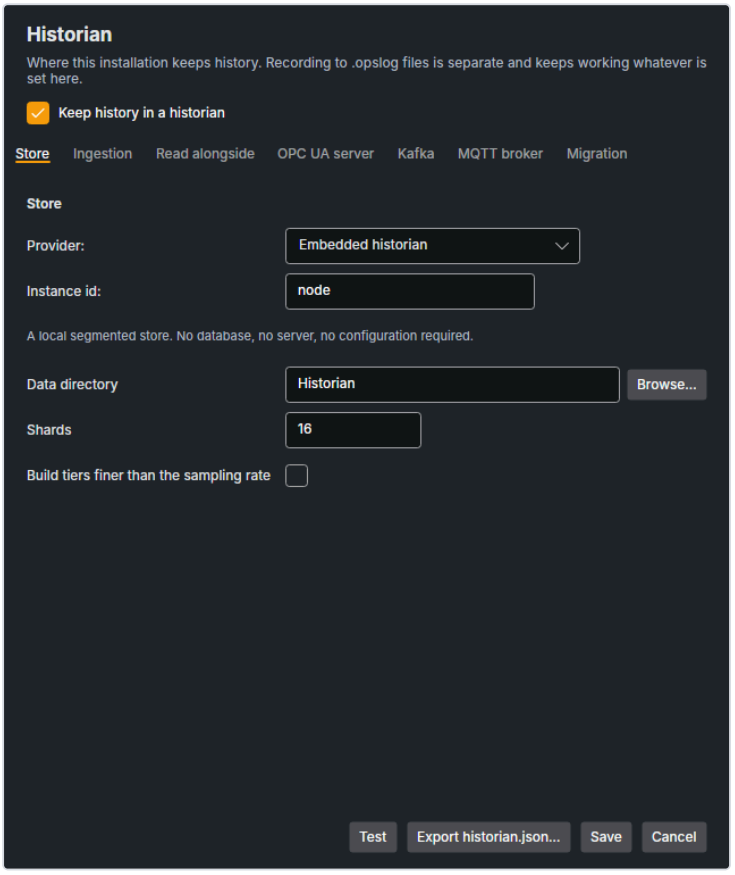
Choosing SQL Server as the store

The historian is configured in two places that write the same setting on the node:

- On the Desktop: **Data ▸ Historian…**
- In the browser, with the Web HMI module: **Administration ▸ Historian**

Either one stores the choice as a configuration revision at the node — numbered, diffable and rollable back.

On the Desktop



The Desktop's Historian settings window, Store tab, with Keep history in a historian ticked. It is shown here with the embedded store selected; choosing Microsoft SQL Server in Provider replaces the Data directory and Shards rows with the SQL Server connection fields

The window has a check box at the top and four tabs:

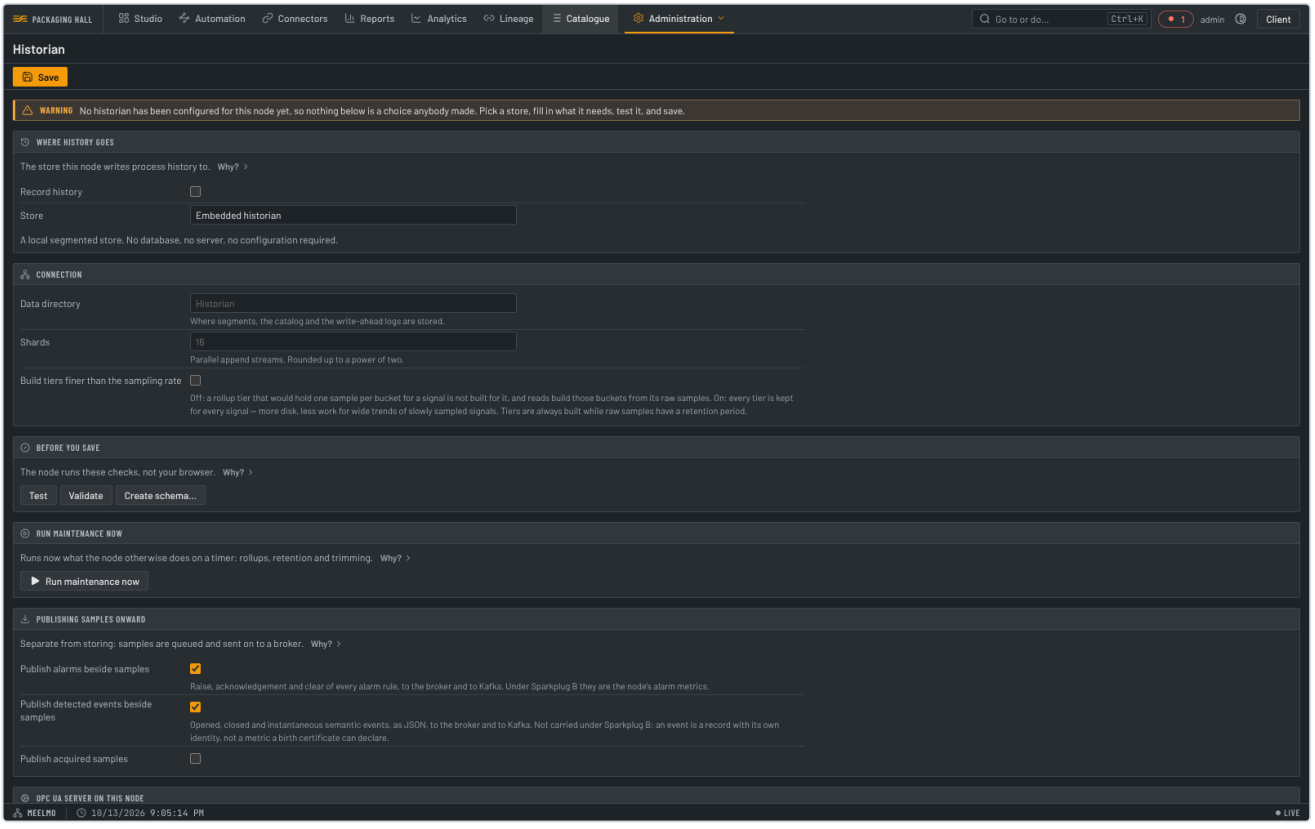
Element	Purpose
Keep history in a historian	The historian is off until this is ticked. Recording to `.opslog` files is separate and keeps working whatever is set here.
Store tab	Which provider, its instance id, and the provider's own connection fields.
Ingestion tab	Queue, batching, the spool, retention, maintenance, rollups and where the semantic model lives.
Read alongside tab	Detection rules file, other stores queried together with the historian, and publication to a broker.
Migration tab	Writing to a second store during a cutover, and the reporting views.
Test	Asks the node to reach the database with the values on the screen.
Create schema	Appears when **Test** found the database reachable but the tables missing.
Export historian.json…	Writes the configuration to a file for a data-access server or web host.
Save / **Cancel**	Save writes every tab at once.

On the **Store** tab:

Field	Meaning
Provider:	Choose **Microsoft SQL Server**. The line under it reads *Stores history in SQL Server using bulk-copy ingestion and typed sample tables.* If the provider is refused (licence or module), the reason is shown here in red instead.
Instance id:	Names this historian instance. It appears in provenance and in migration checkpoints, so two instances holding different data must not share a name. A node's own store is `node`.

The connection fields below are generated from what the SQL Server provider declares, so the Desktop and the browser always offer the same fields. A required field is marked `*`; hovering over a field shows its description.

In the browser



Administration ▸ Historian in the browser before any historian has been configured. The warning banner says nothing below is a choice anybody made yet; the panels are Where history goes, Connection, Before you save, Run maintenance now and Publishing samples onward

The browser page needs an administrator. It has:

Panel	Contents
Toolbar	**Save** — stores the configuration as a revision at the node.
Where history goes	**Record history** (the on/off switch) and **Store** (the provider list). The provider's description is shown under the list. A provider that is not licensed or not enabled is listed as *not available here* and cannot be selected.
Connection	The provider's generated fields (below).
Before you save	**Test**, **Validate** and **Create schema…**. The node runs these, not your browser.
Run maintenance now	Builds rollups, applies retention and trims the semantic model immediately instead of waiting for the timer.

Panel	Contents
Publishing samples onward	Publication to an MQTT broker; covered by the industrial integration module where it is licensed.

The browser page does not offer the **Ingestion**, **Read alongside** or **Migration** settings; those are set on the Desktop.

Connection fields

Field	Required	Default	What to enter
Connection string	Yes	—	Server, database and authentication, in ordinary SQL Server connection-string form. `Encrypt=True` is added when the string does not mention `Encrypt`. Keep the password out of it — use the field below.
Password	No	—	Stored through the secret protector and merged into the connection when the node connects. It is sealed on the node and never displayed again.
Schema	No	`meelmo`	The database schema that holds the tables. Must be a plain identifier.
Bulk batch size	No	`10000`	Rows per bulk-copy operation. Clamped to 100–500,000.
Command timeout (s)	No	`120`	Seconds a single command may run. Clamped to 5–3,600. Retention and rollup commands are allowed four times this.

An empty **Password** box that shows *stored — leave blank to keep it* (Desktop) or *unchanged — type to replace* (browser) means a password is already sealed on the node; leave it empty to keep it, type to replace it. A password is never sent back to the screen.

Numeric fields show their default as a grey hint. Leaving a field empty uses the default; typing the default value records that you chose it.

Example connection strings:

```
Server=sql01.plant.local;Database=MeelmoHistory;User Id=meelmo_writer;TrustServerCertificate=False
Server=sql01.plant.local\HIST;Database=MeelmoHistory;Integrated Security=True
```

With `Integrated Security=True` the node connects as the Windows account it runs under; the **Password** field is then left empty.

The product sets the connection's application name to `Meelmo Historian`, so the DBA can find the historian's sessions in the server's activity views.

Test, Validate and Create schema

These three actions run **at the node**, with the node's providers, the node's network and the credentials in the form. A workstation that is not attached to a node says so rather than answering from its own machine.

Action	What it does	Changes the database?
Test	Opens a connection with the values on screen, and reports whether the schema is ready.	No

Action	What it does	Changes the database?
Validate	Checks that every required table exists in the schema and that the account has `SELECT`, `INSERT` and `DELETE` where the historian needs them.	No
Create schema…	Issues the DDL that creates or upgrades the tables. Asks for confirmation first.	Yes

Create schema is idempotent: running it against a database that already has the schema is a no-op, not an error. It is not undone by rolling back a configuration revision — the tables stay. It is a separate action from **Test** on purpose, so building a schema can never happen as a side effect of asking a question.

The results you can see:

Message	Meaning	Next step
Connection succeeded.	The node reached SQL Server and the schema is ready.	Save.
The store answered, but its schema is not ready: … (Desktop) / *The database is reachable and the tables are not there yet. Create schema builds them.* (browser)	The login works; the tables are missing.	Press **Create schema**.
Connection failed: …	The node could not open a connection. The server's own message follows.	Check server name, port, firewall, login and password from the node's machine.
The schema is present and the account has what it needs.	**Validate** passed.	Save.
Table [meelmo].[OA_…] is missing.	Validate found an incomplete schema.	**Create schema** (it upgrades).
The configured account lacks INSERT on [meelmo].[OA_SAMPLES_NUMERIC].	A grant is missing.	Ask the DBA for the grant named.

The tables it creates

All names begin with `OA_` and are generic, so your own SQL never depends on a process-specific word:

Table	Holds
`OA_SCHEMA_VERSION`	The schema version this build created.
`OA_SIGNALS`, `OA_ASSETS`	The signal catalogue and the asset hierarchy.
`OA_SAMPLES_NUMERIC`, `OA_SAMPLES_INTEGER`, `OA_SAMPLES_BOOLEAN`, `OA_SAMPLES_TEXT`	Raw samples, one table per value type.
`OA_ROLLUP_1S`, `OA_ROLLUP_1M`	One-second and one-minute aggregates.
`OA_INGEST_STATUS`	What each batch delivered.
`OA_EVENTS`, `OA_EXECUTIONS`, `OA_STATES`, `OA_CONTEXT_VALUES`, `OA_RELATIONSHIPS`	The semantic model, when it is stored in the database.

Do not write reports against these tables. Their shape may change between versions; the `oa_v_*` views below will not.

Worked example: moving the packaging line to SQL Server

The demonstration packaging line has one connection, `Packaging`, and signals such as `Packaging/Checkweigher/CaseWeight`, `Packaging/Loader/Rate` and `Packaging/Line/GoodCount`. This example puts its history into a database called `MeelmoHistory` on the server `sql01`.

The DBA has created the database, a login `meelmo_admin` with `CREATE TABLE`, `ALTER` and `CREATE VIEW` on schema `meelmo`, and a login `meelmo_writer` with the runtime grants above.

1. Sign in to the Desktop as `admin` and open **Data ▸ Historian…**.
2. Tick **Keep history in a historian**.
3. On the **Store** tab, choose **Microsoft SQL Server** in **Provider:**. Leave **Instance id:** as `node`.
4. In **Connection string** enter `Server=sql01.plant.local;Database=MeelmoHistory;User Id=meelmo_admin`.
5. In **Password** type the `meelmo_admin` password.
6. Leave **Schema** empty (it will be `meelmo`), **Bulk batch size** empty (10,000) and **Command timeout (s)** empty (120).
7. Press **Test**. The message reads *The store answered, but its schema is not ready*, and **Create schema** appears.
8. Press **Create schema** and confirm. The message reads *Schema created and checked.*
9. On the **Migration** tab, press **Check views**. The node answers that no reporting views exist, and **Create views…** appears. Press it and confirm, so reporting can start straight away.
10. Back on the **Store** tab, change the connection string to `User Id=meelmo_writer` and type the `meelmo_writer` password.
11. Press **Test** again. It now reads *Connection succeeded.*
12. On the **Ingestion** tab, set **Raw retention (days):** to `90` and leave **Build rollups for faster long-range queries** ticked (see *Retention* below).
13. Press **Save**. The Desktop reports that the historian is now writing to the store you chose, that acquisition was not interrupted, and that history already recorded stays in the store that recorded it.

The same steps in the browser: **Administration ▸ Historian**, tick **Record history**, choose **Microsoft SQL Server** in **Store**, fill **Connection**, press **Test**, then **Create schema…** and **Create it**, then **Validate**, then **Save**. The banner reads *Saved. History will go to Microsoft SQL Server.*

Within a minute of saving, `SELECT COUNT(*) FROM meelmo.oa_v_samples WHERE signal_id = 'Packaging/Checkweigher/CaseWeight'` returns a growing number.

Retention and rollups

Retention and rollups are set on the Desktop's **Ingestion** tab under **Maintenance**, and apply to whichever store is chosen.

Field	Default	Meaning
Raw retention (days):	`0`	Days of raw samples to keep. `0` keeps everything.
Maintenance interval (min):	`60`	How often rollups and retention run.
Build rollups for faster long-range queries	On	Builds the one-second and one-minute tiers. Without them, a chart over a month reads raw samples.
Context store:	`journal`	Where the semantic model lives: `journal` (a checksummed journal beside the history, answered from memory) or `provider` (tables in this SQL Server database, in the same backup and the same `SELECT` as the samples).

How SQL Server applies them:

- **Rollups** are built into `OA_ROLLUP_1S` and `OA_ROLLUP_1M` with `MERGE`, so a second pass refreshes a bucket rather than skipping it, and a late sample is folded into its bucket on the next pass. A bucket is not published until the next one has started, so it is not written while samples may still be arriving inside it. Each bucket holds the count, minimum and maximum with their instants, first and last value, sum, sum of squares, and good and bad counts.
- **Retention** deletes raw samples older than the cutoff from the four sample tables, in chunks of 50,000 rows, so the transaction log and the lock footprint stay bounded and ingestion is never stalled behind one huge `DELETE`. Rollups are kept. Nothing newer than the cutoff is removed.
- **Timing.** The node waits one full interval before its first pass. On a fresh installation there is therefore nothing to see in the rollup tables for an hour. **Run maintenance now** (browser) runs a pass immediately.

A typical choice for the packaging line: 90 days of raw samples, rollups kept indefinitely. A chart over a quarter then reads the one-minute tier, and a drill-down into last week still has every sample.

Ingestion and the spool

The **Ingestion** tab controls how samples travel from acquisition to SQL Server.

Field	Default	Meaning
Queue capacity:	`200000`	Samples the in-memory queue holds before it refuses more. At least 1,000.
Batch size:	`5000`	Samples per write to the provider.
Flush interval (ms):	`500`	How long a partial batch waits before being sent anyway.
Spool folder:	`spool`	Where samples wait on disk while SQL Server is unreachable. Relative to the data directory.
Spool limit (GB):	`8`	Ceiling on the spool.
When the spool is full:	`StopAccepting`	`StopAccepting` refuses new samples for this store and counts them; `OverwriteOldest` discards the oldest spooled data.

When SQL Server is restarting, failing over, refusing connections or refusing the login, the batch is **spooled** and replayed when the server returns. Acquisition, live values, recording and the embedded store keep running. A batch the server accepts the connection for but rejects as data — a value too large for its column, a schema that does not match — is written to a dead-letter file beside the spool (`dead-letter.dat` and `dead-letter.log`) rather than thrown away, and the historian status names the file until somebody deals with it.

> **Keep the spool at 8 GB, and never below 64 MB.** Below that the cap cannot be enforced correctly and a spool that fills once will not recover. Back up the spool folder with the historian.

The historian reports one of four states — Disabled, Running, Degraded (spooling) and Unavailable — in the Desktop's status bar and in the node's diagnostics.

Read alongside and migration from the embedded historian

A site that started on the embedded store and buys this module usually wants two things: new samples in SQL Server from now on, and the old history still visible. There are three tools for that, and they combine.

Keep the old store readable

On the **Read alongside** tab, **Add…** opens *Read alongside the historian*. It adds a read-only store that is queried together with the historian, so a trend across the change-over day shows both halves as one line.

Field	Meaning
Reads:	*Recordings (.opslog files)*, a Meelmo store kept after a migration, or *A database this product did not create*.
Id:	Required. Travels with every point the store serves, so a chart can say where a value came from.
Name:	A display name.
Priority:	Where two stores hold the same instant of the same signal, the lower number wins. The historian itself is `0`.
Include this store in queries	Untick to keep the entry but stop reading it.

Nothing listed on this tab is ever written to.

Write to both during a cutover

On the **Migration** tab, **Also write to a second store, for a cutover** sends every batch to a second store while older history is back-filled.

Field	Meaning
Second store:	The provider being introduced.
Second instance id:	Must differ from the historian's own (default `migration-target`).
Connection string: / **Password:**	The second store's connection. The password is sealed, never written into the settings file.

The store chosen on the **Store** tab stays in charge. The second store is best-effort: what it does not take is counted rather than retried forever, so you can see whether it is behind before cutting over.

Back-fill the history: `Meelmo.Migrate`

`Meelmo.Migrate` copies the history already held into SQL Server, oldest first, in resumable windows, after profiling the import and refusing it on evidence. The program, its configuration files, every option, the refusal codes and the exit codes are described once, for every store, in *The historian*. What is particular to SQL Server:

- **The target file.** `"providerId": "sqlserver"`, with the settings `connectionString`, `schema`, `bulkBatchSize` and `commandTimeoutSeconds`, and the password as a sealed token under `secretRefs` with the key `password`. A sealed token opens only on the machine that sealed it, so run the tool on the machine that holds the historian configuration.
- **Use a checkpoint.** The SQL Server store does not deduplicate a re-sent sample, so an interrupted run started again without `--checkpoint` would send everything it had already copied once more. The profile notes `no-checkpoint`, and the run is refused unless you say `--allow-rerun-without-dedup`.
- **The schema.** A database administrator normally creates the tables with **Create schema** (see *Test, Validate and Create schema* above). `--create-schema` creates them instead with the credentials in the target file, which then need the rights to do it. Without either, the tool refuses a target whose schema is not ready.

- **Into the live store, set a ceiling.** When SQL Server is already the store the node acquires into, `--max-rate` keeps the node's own writing ahead of the back-fill.

```
Meelmo.Migrate --source embedded.json --target sqlserver.json ^
             --checkpoint C:\ProgramData\Meelmo\migrate\packaging.ckpt --dry-run
```

The recommended order

1. Configure SQL Server as the **second store** on the **Migration** tab and save. New samples now go to both.
2. Run `Meelmo.Migrate --dry-run`, then without `--dry-run`, from the embedded store into SQL Server with a checkpoint and `--to` set to the moment you saved in step 1. Re-run with the same checkpoint until the report shows no failed windows. The embedded store can be read only while no node holds it (see *Moving history between stores*).
3. Check that the second store reports nothing lost.
4. On the **Store** tab, make SQL Server the historian; untick the second store; add the embedded store under **Read alongside** if you want to keep reading it rather than delete it. Save.

Reporting views

With the direct-database-access entitlement, the **Migration** tab's **Reporting views** section creates seven stable views for your own SQL:

View	Holds
`oa_v_signals`	The catalogue: `signal_id`, display name, type, unit, asset path
`oa_v_samples`	Raw samples: `signal_id`, `source_time_utc`, `quality`, `numeric_value`, `text_value`
`oa_v_rollup_1s`, `oa_v_rollup_1m`	Aggregates: `bucket_utc`, count, min, max, first, last, mean, good and bad counts
`oa_v_events`, `oa_v_executions`, `oa_v_states`	The semantic model

Check views asks the node which exist; after it has answered, **Create views…** appears, and **Remove views…** appears when there are views to remove. The node builds them with the credentials this configuration names, which need `CREATE VIEW`. On success the message lists the views and reminds you to grant `SELECT` on them only. Grant your reporting login `SELECT` on these views only, never on the tables underneath.

```
-- Hourly average case weight on the packaging line, last 7 days
SELECT DATEADD(hour, DATEDIFF(hour, 0, bucket_utc), 0) AS hour_utc,
       AVG(mean_value) AS avg_case_weight
FROM meelmo.oa_v_rollup_1m
WHERE signal_id = 'Packaging/Checkweigher/CaseWeight'
  AND bucket_utc >= DATEADD(day, -7, SYSUTCDATETIME())
GROUP BY DATEADD(hour, DATEDIFF(hour, 0, bucket_utc), 0)
ORDER BY hour_utc;
```

Always filter by `signal_id` and by time. Once your own login queries the database directly, Meelmo is not in the path and cannot apply per-user signal scoping; grant that access as you would any other database access.

Sizing

Order of magnitude, raw samples before retention and deadband:

Tags	Rate	Raw per day	Raw per year
1,000	1 s	~2.7 GB	~1.0 TB
1,000	100 ms	~27 GB	~10 TB
10,000	1 s	~27 GB	~10 TB
10,000	100 ms	~273 GB	~100 TB

For the packaging line's roughly 140 signals at one sample a second, that is about 0.4 GB a day. With 90 days of raw retention the sample tables level off at around 35 GB, and the rollup tables grow far more slowly after that.

Two things move these numbers more than the table suggests: **retention**, and **deadband** — most process tags do not change every scan.

Spool sizing at the default 8 GB: about 30 days of outage at 100 samples a second, about 3 days at 1,000, about 7.5 hours at 10,000.

For SQL Server itself:

- Put the database in the recovery model your DBA uses for other operational data; retention's chunked deletes keep each transaction small either way.
- Size the transaction log for one maintenance pass, not for the whole history.
- Raise **Bulk batch size** only if the server shows many small bulk inserts; the default suits most sites.

Troubleshooting

Symptom	Cause and remedy
Microsoft SQL Server is greyed out, *not available here*.	The licence lacks the SQL Server capability or `historian-sqlserver` is switched off. Install or enable the module under **Administration ▸ Modules**.
The Desktop says the provider is *not installed on this machine*.	Run the installer again and tick **Database history providers**.
Test says *This installation is not attached to a node*.	The Desktop has no node to ask. Configure the node connection first.
Connection failed with a network error.	The node cannot reach the server. Test from the node's own machine: name resolution, port 1433 (or the instance port), firewall.
Connection failed with a certificate error.	Encryption is on by default. Install a certificate the node trusts on SQL Server, or state `Encrypt=` explicitly in the connection string if your site has decided otherwise.
'...' is not a valid SQL Server schema name.	**Schema** must be a plain identifier such as `meelmo` or `hist`.
After saving, *The node could not open that store, so this host is still writing to the one it had.*	The setting is saved; the node kept its previous store. Fix what the message names and save again, or restart the host.

Symptom	Cause and remedy
The historian shows Degraded (spooling).	SQL Server is unreachable. Samples are being kept on disk and will replay. Watch the spool's size against its limit.
The status names a dead-letter file.	SQL Server rejected a batch as data. Read `dead-letter.log` for the server's reason, fix it (usually a schema mismatch: press **Create schema** to upgrade), and keep the file until it has been replayed.
Rollup tables are empty an hour after installation.	The first maintenance pass waits one full interval. Use **Run maintenance now**, or wait.
Create views… fails with a permissions error.	The account in the form needs `CREATE VIEW` on the schema — the same account that created the tables.

Oracle historian

> **Module: Oracle historian (`historian-oracle`).** This chapter describes the Oracle Database store for the historian. It is present only on installations licensed for that module. Everything about the historian that does not depend on the database — what is recorded, rollups, the history browser, the spool — is in the base chapter *The historian*; this chapter adds what is specific to keeping that history in Oracle.

What this module adds

The base product keeps history in the embedded store or in a SQLite file. This module adds an Oracle Database that your own DBAs run, back up and report against.

Choose it when the site's operational data already lives in Oracle, when reporting tools should read plant history through SQL, or when history must be kept for years under the IT department's storage, backup and security regime. Keep the embedded store when none of that applies.

What the Oracle store does, in short:

Aspect	Behaviour
Ingestion	Array binding: one statement carries thousands of rows. No statement is ever issued per sample.
Tables	Typed, index-organized sample tables — numeric, integer, boolean and text — keyed on signal, source time and sequence; index-organized one-second and one-minute rollups; the signal catalogue and the semantic model.
Ownership	The tables may belong to a schema owner other than the account the historian connects as — the normal arrangement for a least-privileged runtime account.
Compression	Not used. Advanced Compression is a separately licensed Oracle option and the product does not assume you have it.
Duplicates	A re-sent identical sample is rejected by the primary key rather than silently deduplicated. This matters when re-running a migration (use a checkpoint).
Reporting	Seven stable `oa_v_*` views for your own SQL.

Before you start

1. **The module, licensed and switched on.** The licence must grant the Oracle provider capability, and `historian-oracle` must be enabled on the node under **Administration ▸ Modules**. Until both hold, **Oracle Database** is listed in the store list but marked *not available here*, with the reason.
2. **The provider installed on the node's machine.** The database stores are an optional installer component. If the Desktop says the provider is *not installed on this machine*, run the installer again and tick **Database history providers**.
3. **A supported database.** Oracle Database 19c or later.
4. **Users and a tablespace.** Ask your DBA for a schema owner (for example `meelmo`) with a quota on a tablespace, a runtime user (`meelmo_writer`), and optionally a reporting user (`meelmo_reader`).
5. **Oracle Net access from the node.** The node opens the connection, not the Desktop and not the browser.

Privileges

User	Used for	Privileges
Schema owner (once)	**Create schema** and the reporting views	`CREATE SESSION`, `CREATE TABLE`, `CREATE SEQUENCE`, and `CREATE VIEW` for the views; a tablespace quota
Runtime user	Writing, reading, retention	`CREATE SESSION`; `SELECT`, `INSERT`, `UPDATE`, `DELETE` on every `OA_` table
Reporting user	Your own SQL	`CREATE SESSION`; `SELECT` on the `oa_v_*` views

The product's starting point (names are examples; repeat the object grants for every `OA_` table):

```
-- Owner, used once to create the schema. Not the service account.
CREATE USER meelmo IDENTIFIED BY "<password>"
    DEFAULT TABLESPACE meelmo_data QUOTA UNLIMITED ON meelmo_data;
GRANT CREATE SESSION, CREATE TABLE, CREATE SEQUENCE TO meelmo;

-- Runtime account: ingestion and queries, no DDL. DELETE is for retention.
CREATE USER meelmo_writer IDENTIFIED BY "<password>";
GRANT CREATE SESSION TO meelmo_writer;
GRANT SELECT, INSERT, UPDATE, DELETE ON meelmo.OA_SIGNALS TO meelmo_writer;
-- ...and the same for every OA_ table.

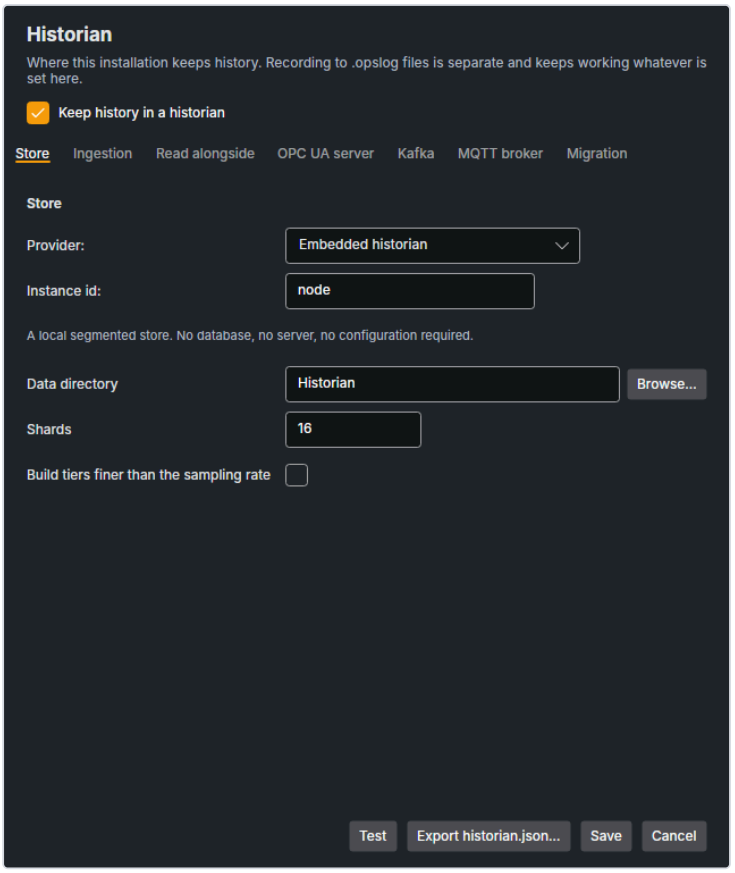
-- Read-only account for reporting and direct SQL access.
CREATE USER meelmo_reader IDENTIFIED BY "<password>";
GRANT CREATE SESSION TO meelmo_reader;
```

Object grants can only be given once the tables exist, so the order is: create the schema as the owner, then grant to the runtime user, then switch the historian to the runtime user.

Choosing Oracle as the store

The historian is configured on the Desktop under **Data ▸ Historian…**, or in the browser, with the Web HMI module, under **Administration ▸ Historian**. Both store the choice as a configuration revision at the node.

On the Desktop

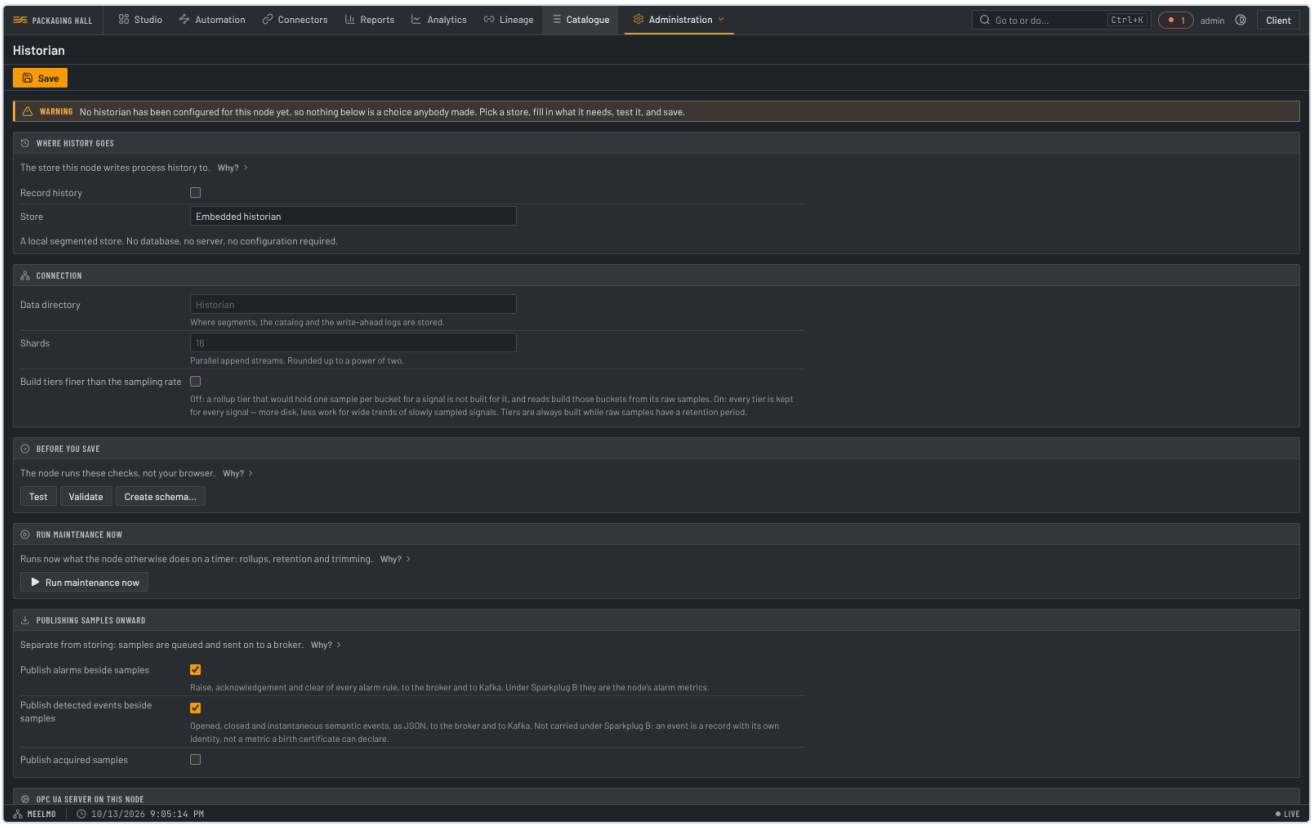


The Desktop's Historian settings window on its Store tab, shown with the embedded store selected. Choosing Oracle Database in Provider replaces the Data directory and Shards rows with the Oracle connection fields

Element	Purpose
Keep history in a historian	The historian is off until this is ticked. Recording to `.opslog` files is independent of it.
Store tab	**Provider:**, **Instance id:** and the provider's connection fields.
Ingestion tab	Queue, batching, spool, retention, maintenance interval, rollups, context store.
Read alongside tab	Detection rules file, read-only stores queried with the historian, and publication.
Migration tab	A second store written during a cutover; the reporting views.
Test	Asks the node to reach the database with the values on screen.
Create schema	Appears when **Test** found the database reachable and the tables missing.
Export historian.json..., **Save**, **Cancel**	Export the configuration; save every tab; close without saving.

Choose **Oracle Database** in **Provider:**. The line below reads *Stores history in Oracle using array-bound ingestion and typed, index-organized sample tables.* **Instance id:** names this historian instance in provenance and migration checkpoints; a node's own store is `node`.

In the browser



Administration ▸ Historian in the browser before a historian has been configured: Where history goes, Connection, Before you save with Test, Validate and Create schema, Run maintenance now, and Publishing samples onward

Tick **Record history**, choose **Oracle Database** in **Store**, fill the **Connection** panel, and use **Test**, **Validate** and **Create schema…** in **Before you save** before pressing **Save** on the toolbar. **Run maintenance now** runs a rollup and retention pass immediately. The ingestion, read-alongside and migration settings are on the Desktop only.

Connection fields

Field	Required	Default	What to enter
Connection string	Yes	—	Data source and user. The password is held separately, never here.
Password	No	—	Sealed on the node and merged into the connection at connect time. Never displayed again.
Schema owner	No	(empty)	The user that owns the tables. Empty means the connecting user's own schema. Must be a plain identifier.
Array batch size	No	`10000`	Rows per array-bound statement. Clamped to 100–500,000.
Command timeout (s)	No	`120`	Seconds a single command may run. Clamped to 5–3,600.

An empty **Password** box showing *stored — leave blank to keep it* (Desktop) or *unchanged — type to replace* (browser) keeps the password already sealed on the node.

Example connection strings:

```
Data Source=ora01.plant.local:1521/MEELMOPDB;User Id=meelmo_writer
Data Source=MEELMOHIST;User Id=meelmo_writer
```

The second form uses a net service name that the node's Oracle client configuration resolves.

Encryption of the Oracle Net connection is configured the way your site already configures it for Oracle clients; the product does not add a setting of its own.

Schema owner, and who the historian connects as

Arrangement	Connection string user	Schema owner
One user owns and writes (simplest, not least-privileged)	`meelmo`	empty
Owner creates, runtime user writes (recommended)	`meelmo_writer`	`meelmo`

With a separate owner, the historian reads and writes `meelmo.OA_…` through the grants. **Validate** checks the grants only when the tables belong to someone else: owning a table carries its privileges without a grant row.

Test, Validate and Create schema

All three run at the node, with the node's network and the credentials in the form.

Action	What it does	Changes the database?
Test	Connects, and reports whether the schema is ready.	No
Validate	Checks every required table in the owner's schema, the schema version, and that the connecting user holds `INSERT` and `DELETE` on the numeric sample table.	No
Create schema…	Creates or upgrades the tables. Asks first.	Yes

Create schema is idempotent and is not undone by rolling back a configuration revision.

Message	Meaning
Schema and privileges are as expected.	**Validate** passed.
Table OA_… is missing. followed by *Run schema creation as a user with CREATE TABLE before starting the historian.*	The schema is absent or incomplete.
The schema is at version N; this build expects M.	The tables predate this version. **Create schema** upgrades them.
The connecting user has no INSERT privilege on MEELMO.OA_SAMPLES_NUMERIC.	A grant is missing for the runtime user.
ORA-nnnnn: …	Oracle's own error, passed through unchanged.
'…' is not a valid Oracle schema owner name.	**Schema owner** must be a plain identifier.

The tables are `OA_SCHEMA_VERSION`, `OA_SIGNALS`, `OA_ASSETS`, `OA_SAMPLES_NUMERIC`, `OA_SAMPLES_INTEGER`, `OA_SAMPLES_BOOLEAN`, `OA_SAMPLES_TEXT`, `OA_ROLLUP_1S`, `OA_ROLLUP_1M`, `OA_INGEST_STATUS`, and the semantic tables `OA_EVENTS`, `OA_EXECUTIONS`, `OA_STATES`, `OA_CONTEXT_VALUES` and `OA_RELATIONSHIPS`. Report against the views, not these.

Worked example: the packaging line on Oracle

The packaging line (connection `Packaging`, signals such as `Packaging/Checkweigher/CaseWeight` and `Packaging/Erector/MagazineLevel`) is to write into the pluggable database `MEELMOPDB` on `ora01`. The DBA has created `meelmo` and `meelmo_writer` as above.

1. On the Desktop, signed in as `admin`, open **Data ▸ Historian…** and tick **Keep history in a historian**.
2. **Store** tab: **Provider:** `Oracle Database`; **Instance id:** `node`.
3. **Connection string:** `Data Source=ora01.plant.local:1521/MEELMOPDB;User Id=meelmo`. **Password:** the owner's password. Leave **Schema owner** empty.
4. Press **Test**. It reports the schema is not ready, and **Create schema** appears. Press it and confirm. The message reads *Schema created and checked.*
5. On the **Migration** tab press **Check views**, then **Create views…**, and confirm.
6. Ask the DBA to run the object grants for `meelmo_writer` on every `OA_` table, and `SELECT` on the `oa_v_*` views for `meelmo_reader`.
7. Back on **Store**: **Connection string:** `Data Source=ora01.plant.local:1521/MEELMOPDB;User Id=meelmo_writer`, the `meelmo_writer` password, and **Schema owner:** `meelmo`.
8. Press **Test**; it reads *Connection succeeded.*
9. **Ingestion** tab: **Raw retention (days):** `180`; leave **Build rollups for faster long-range queries** ticked.
10. Press **Save**.

In the browser the equivalent is **Administration ▸ Historian**: **Record history**, **Store** `Oracle Database`, the three connection values, **Test**, **Validate** (which should read *Schema and privileges are as expected.*), then **Save**.

Retention and rollups

Set on the Desktop's **Ingestion** tab under **Maintenance**:

Field	Default	Meaning
Raw retention (days):	`0`	Days of raw samples to keep; `0` keeps everything.
Maintenance interval (min):	`60`	How often rollups and retention run.
Build rollups for faster long-range queries	On	One-second and one-minute tiers.
Context store:	`journal`	`journal`, or `provider` to keep the semantic model in the Oracle tables beside the samples.

On Oracle:

- **Rollups** go into the index-organized `OA_ROLLUP_1S` and `OA_ROLLUP_1M` tables. A pass refreshes buckets it has already written, so a late sample is folded in on the next pass, and a bucket is not published while samples may still arrive inside it.
- **Retention** deletes raw samples older than the cutoff in bounded chunks of 50,000 rows per statement, rather than one `DELETE` over a year of samples that would hold undo for as long as it runs. Rollups are kept.
- **First pass.** The node waits a full maintenance interval before its first pass; **Run maintenance now** in the browser runs one immediately.

Ingestion and the spool

The **Ingestion** tab's **Queue capacity:** (200,000), **Batch size:** (5,000), **Flush interval (ms):** (500), **Spool folder:** (`spool`), **Spool limit (GB):** (8) and **When the spool is full:** (`StopAccepting` or `OverwriteOldest`) behave exactly as for every store.

An Oracle outage — the listener down, the database restarting, the login refused — is spooled to disk and replayed when the database returns; acquisition, live values and recording carry on. A batch Oracle accepts the connection for but rejects as data is kept in `dead-letter.dat` / `dead-letter.log` beside the spool, with Oracle's reason, and the historian status names the file.

> **Keep the spool at 8 GB, and never below 64 MB.** Back up the spool folder with the historian.

Read alongside and migration from the embedded historian

The tools are the same for every server store.

Keep the old store readable. Read alongside tab, **Add…**: choose what the source **Reads:** (recordings, a Meelmo store kept after a migration, or a database this product did not create), give it an **Id:**, a **Name:** and a **Priority:** (the historian is `0`; lower wins where two stores hold the same instant), and tick **Include this store in queries**. Nothing listed here is written to.

Write to both during the cutover. Migration tab, **Also write to a second store, for a cutover**: **Second store:** `Oracle Database`, **Second instance id:** (default `migration-target`), **Connection string:** and **Password:**. The first store stays in charge; the second is best-effort and counts what it misses.

Back-fill with `Meelmo.Migrate`. The program, its configuration files, every option, the refusal codes and the exit codes are described once, for every store, in *The historian*. What is particular to Oracle:

- **The target file.** `"providerId": "oracle"`, with the settings `connectionString`, `owner`, `arrayBatchSize` and `commandTimeoutSeconds`, and the password as a sealed token under `secretRefs` with the key `password`. Run the tool on the machine that sealed it.
- **Always use `--checkpoint`.** Oracle rejects an exact duplicate rather than recognising it, so an interrupted run re-sent without a checkpoint would fail windows it had already copied. Without a checkpoint the run is refused unless you say `--allow-rerun-without-dedup`, which is reasonable only for a first run into an empty store.
- **The schema.** Create it first with **Create schema** (see *Test, Validate and Create schema* above), in the recommended arrangement as the schema owner. `--create-schema` would create it with the credentials in the target file instead, which are usually the runtime user's. Without either, the tool refuses a target whose schema is not ready.

```
Meelmo.Migrate --source embedded.json --target oracle.json ^
            --checkpoint C:\ProgramData\Meelmo\migrate\packaging-oracle.ckpt --dry-run
```

The recommended order is: second store on, save; back-fill with a checkpoint, and with `--to` set to the moment you saved, until the report shows no failed windows; confirm the second store lost nothing; make Oracle the historian, turn the second store off, and keep the old store under **Read alongside** if it should stay readable.

Reporting views

With the direct-database-access entitlement, the **Migration** tab's **Reporting views** section manages seven views in the owner's schema: `oa_v_signals`, `oa_v_samples`, `oa_v_rollup_1s`, `oa_v_rollup_1m`, `oa_v_events`, `oa_v_executions` and `oa_v_states`. **Check views** asks the node which exist; **Create views…** and **Remove views…** appear after it has answered. The account in the form needs `CREATE VIEW`.

```
-- Hourly mean case weight, last 7 days
SELECT TRUNC(bucket_utc, 'HH24') AS hour_utc,
       AVG(mean_value)           AS avg_case_weight
FROM meelmo.oa_v_rollup_1m
WHERE signal_id = 'Packaging/Checkweigher/CaseWeight'
  AND bucket_utc >= SYS_EXTRACT_UTC(SYSTIMESTAMP) - INTERVAL '7' DAY
GROUP BY TRUNC(bucket_utc, 'HH24')
ORDER BY hour_utc;
```

Grant `SELECT` on the views only, never on the tables. A login reading the database directly bypasses Meelmo's per-user signal scoping.

Sizing

Tags	Rate	Raw per day	Raw per year
1,000	1 s	~2.7 GB	~1.0 TB
1,000	100 ms	~27 GB	~10 TB
10,000	1 s	~27 GB	~10 TB

These are before retention and deadband, and without Oracle compression, which the product does not use. For the packaging line's roughly 140 signals at one sample a second, expect about 0.4 GB of raw samples a day; with 180 days of retention, about 70 GB in the sample tables.

For the database:

- Size the tablespace for the retention window plus the rollups, and the undo tablespace for one chunk of 50,000 deletes per table, not for the whole history.
- Index-organized sample tables store rows in key order, so reads of one signal over a period are contiguous; they also mean each sample's key is its storage.
- Raise **Array batch size** only if the database shows many small array inserts.

Troubleshooting

Symptom	Cause and remedy
Oracle Database is *not available here*.	Licence or module. Enable `historian-oracle` under **Administration ▸ Modules**.
The provider is *not installed on this machine*.	Re-run the installer with **Database history providers** ticked.
Connection failed: ORA-12154 or *ORA-12514*.	The data source cannot be resolved or the service is unknown. Use the `host:port/service` form, or fix the node's net service names.
ORA-01017: invalid username/password.	Retype the password; the stored one is kept only if the box is left empty.

Symptom	Cause and remedy
Table OA_... is missing while the tables exist.	**Schema owner** is empty but the tables belong to another user. Set it to the owner.
The connecting user has no DELETE privilege...	Grant `DELETE` on every `OA_` sample table; retention needs it.
ORA-01950: no privileges on tablespace during **Create schema**.	The owner lacks a tablespace quota.
After saving, *the node could not open that store, so this host is still writing to the one it had*.	The setting is saved; fix what is named and save again, or restart the host.
Degraded (spooling).	Oracle is unreachable; samples are waiting on disk and will replay.
A dead-letter file is named.	Oracle rejected a batch as data; `dead-letter.log` holds its reason. Usually a schema older than the build — **Create schema** upgrades it.
A migration re-run fails windows with unique-constraint errors.	It was re-run without its checkpoint. Re-run with the same `--checkpoint` file.

TimescaleDB historian

> **Module: TimescaleDB historian (`historian-timescale`).** This chapter describes the TimescaleDB store for the historian. It is present only on installations licensed for that module. Everything about the historian that does not depend on the database — what is recorded, rollups, the history browser, the spool — is in the base chapter *The historian*; this chapter adds what is specific to keeping that history in TimescaleDB on PostgreSQL.

What this module adds

TimescaleDB is PostgreSQL with the `timescaledb` extension, which stores time-series in **hypertables**: tables split into time chunks behind the scenes. This module lets the node's historian write there.

Choose it when history must be kept at scale on open-source infrastructure, when Grafana's PostgreSQL data source should read plant history directly, or when the site already runs PostgreSQL. Keep the embedded store when none of that applies.

What the TimescaleDB store does, in short:

Aspect	Behaviour
Ingestion	Binary `COPY` into an unlogged staging table, then one deduplicating insert (`INSERT … ON CONFLICT DO NOTHING`) into the hypertable.
Deduplication	A re-sent sample — same signal, source time and sequence — is recognised and ignored. Replays and re-run migrations do not duplicate data.
Tables	Typed sample hypertables (numeric, integer, boolean, text) chunked by `source_time_utc` in 7-day chunks; one-second and one-minute rollups; the catalogue and the semantic model.
Retention	Drops whole chunks with `drop_chunks` rather than deleting rows.
Encryption	`SSL Mode=VerifyFull` — TLS required and the server's certificate checked — when the connection string does not state an SSL mode. Running without TLS needs **Allow an unencrypted connection**, and is logged as a warning.
Reporting	Seven stable `oa_v_*` views — suitable for Grafana's PostgreSQL data source.

Before you start

1. **The module, licensed and switched on.** The licence must grant the TimescaleDB provider capability and `historian-timescale` must be enabled under **Administration ▸ Modules**. Otherwise **TimescaleDB** is listed but marked *not available here*, with the reason.
2. **The provider installed on the node's machine** — the optional **Database history providers** installer component.
3. **PostgreSQL 14 or later with the TimescaleDB extension installed in the database.** The product checks this at connection time. A provider named *timescale* running against plain PostgreSQL would create tables and then fail on the first hypertable, so it refuses instead: *The database 'meelmo' does not have the timescaledb extension installed. Run CREATE EXTENSION timescaledb; as a superuser, or choose a different provider.*
4. **Roles.** An owner role that creates the schema once, a runtime role for the historian, and optionally a reporting role.
5. **Network access from the node** to PostgreSQL (port 5432 by default).

Privileges

```
-- As a superuser, in the target database:
CREATE EXTENSION IF NOT EXISTS timescaledb;

-- Owner, used once to create the schema. Not the service account.
CREATE ROLE meelmo_admin LOGIN PASSWORD '<password>';
GRANT CREATE ON DATABASE meelmo TO meelmo_admin;

-- Runtime account: ingestion and queries, no DDL. DELETE is for retention.
CREATE ROLE meelmo_writer LOGIN PASSWORD '<password>';
GRANT USAGE ON SCHEMA meelmo TO meelmo_writer;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA meelmo TO meelmo_writer;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA meelmo TO meelmo_writer;

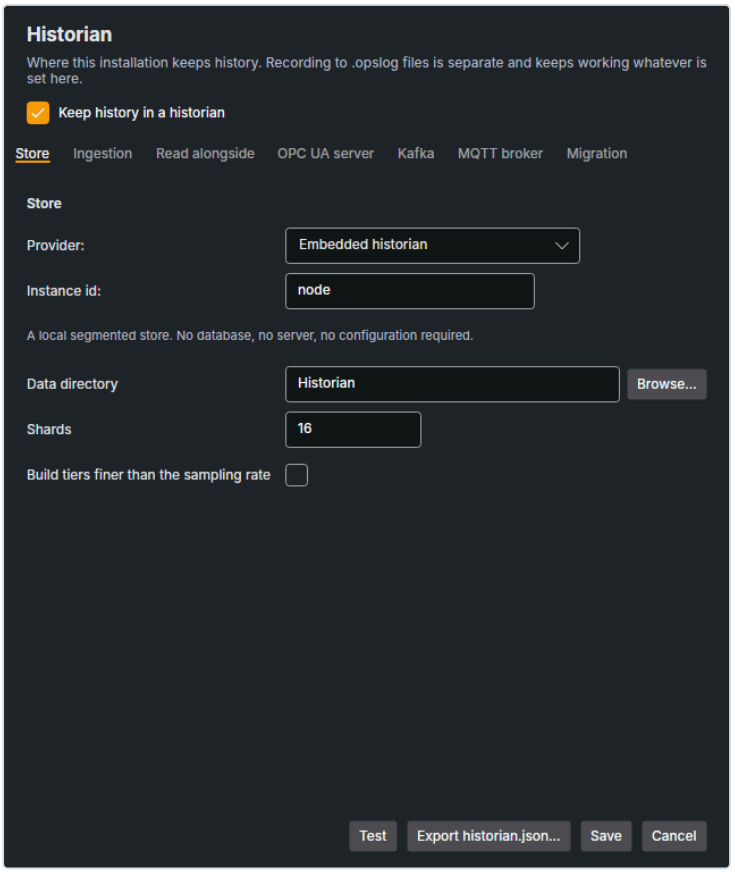
-- Read-only account for reporting and direct SQL access.
CREATE ROLE meelmo_reader LOGIN PASSWORD '<password>';
GRANT USAGE ON SCHEMA meelmo TO meelmo_reader;
GRANT SELECT ON ALL TABLES IN SCHEMA meelmo TO meelmo_reader;
```

The schema-level grants refer to tables that exist, so run the `meelmo_writer` and `meelmo_reader` grants after **Create schema**. Retention calls `drop_chunks` on the sample hypertables, which PostgreSQL permits only to the tables' owner; if retention is configured, either run the historian as the owning role or have your DBA arrange ownership accordingly.

Choosing TimescaleDB as the store

Configure the historian on the Desktop under **Data ▸ Historian…**, or in the browser, with the Web HMI module, under **Administration ▸ Historian**. Both save a configuration revision at the node.

On the Desktop

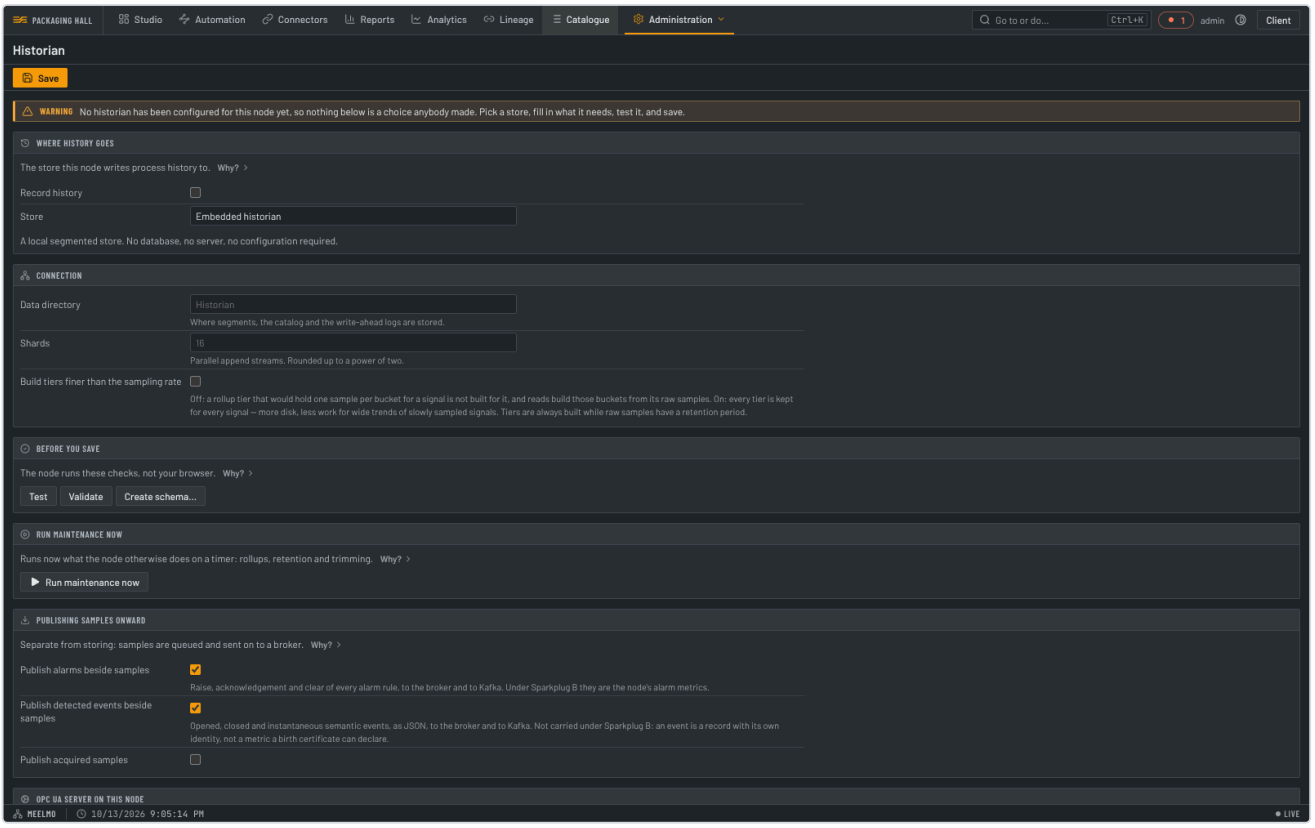


The Desktop's Historian settings window on its Store tab, shown with the embedded store selected. Choosing TimescaleDB in Provider replaces the Data directory and Shards rows with the TimescaleDB connection fields

Element	Purpose
Keep history in a historian	The historian is off until ticked. `.opslog` recording is independent.
Store tab	**Provider:**, **Instance id:**, and the connection fields.
Ingestion tab	Queue, batches, spool, retention, maintenance, rollups, context store.
Read alongside tab	Detection rules, other read-only stores, publication.
Migration tab	A second store for a cutover; reporting views.
Test / **Create schema**	Reach the database; build the tables when **Test** finds them missing.
Export historian.json…, **Save**, **Cancel**	Export; save every tab; close.

Choose **TimescaleDB** in **Provider:**. The line below reads *Stores history in TimescaleDB hypertables using binary COPY with deduplicating inserts.* Leave **Instance id:** as `node` for a node's own store.

In the browser



Administration ▸ Historian in the browser before a historian has been configured: Where history goes, Connection, Before you save, Run maintenance now, and Publishing samples onward

Tick **Record history**, choose **TimescaleDB** in **Store**, fill **Connection**, then **Test**, **Validate**, and if needed **Create schema…**, and press **Save**. **Run maintenance now** runs rollups and retention immediately. Ingestion, read-alongside and migration settings are set on the Desktop.

Connection fields

Field	Required	Default	What to enter
Connection string	Yes	—	Host, database and user, in Npgsql form. The password is held separately. `SSL Mode=VerifyFull` is added when you do not state one.
Password	No	—	Sealed on the node, merged in at connect time, never displayed again.
Schema	No	`meelmo`	The PostgreSQL schema for the tables. A plain identifier.
Copy batch size	No	`50000`	Rows per binary `COPY`. Clamped to 100–1,000,000.
Command timeout (s)	No	`120`	Seconds a command may run. Clamped to 5–3,600.
Allow an unencrypted connection	No	off	Only for a database with no TLS. The password and every sample cross the network in the clear, and a warning is logged at every start. Applies only when the connection string states no SSL mode.

Example connection strings:

```
Host=tsdb01.plant.local;Port=5432;Database=meelmo;Username=meelmo_writer
Host=tsdb01.plant.local;Database=meelmo;Username=meelmo_writer;Root Certificate=/etc/meelmo/pg-
```

```
ca.crt
```

With no SSL mode stated the connection requires TLS and checks the server's certificate against the machine's trusted roots and the host name (`VerifyFull`). For a database whose certificate comes from a private authority, name that authority's certificate with `Root Certificate` , as in the second example. A stated SSL mode is used as written; `Disable` , `Allow` , `Prefer` and `Require` are logged as a warning, because each lets the connection be read or the server be impersonated. Before this release an unstated mode meant `Prefer` ; the *Installation Guide*, §8, has the migration note. The application name on the connection is `Meelmo Historian` , so the historian's sessions are easy to find in `pg_stat_activity` .

Test, Validate and Create schema

All three run at the node.

Action	What it checks or does	Changes the database?
Test	Connects, checks that the `timescaledb` extension is installed, and reports whether the schema is ready.	No
Validate	The extension is installed; every required table exists in the schema; the sample tables are hypertables; the schema version is the one this build expects.	No
Create schema…	Creates or upgrades the tables and turns the sample tables into hypertables. Asks first.	Yes

Message	Meaning
Schema, hypertables and extension are as expected.	**Validate** passed.
The timescaledb extension is not installed in this database.	A superuser must run `CREATE EXTENSION timescaledb;` in that database.
Table oa_… is missing.	Run **Create schema**.
Table oa_samples_numeric exists but is not a hypertable.	The table was created by something else, or without the extension. Consult your DBA before converting it.
The schema is at version N; this build expects M.	**Create schema** upgrades it.
42501: permission denied …	PostgreSQL's own code and message, passed through. A grant is missing.

The tables are lower-case: `oa_schema_version` , `oa_signals` , `oa_assets` , `oa_samples_numeric` , `oa_samples_integer` , `oa_samples_boolean` , `oa_samples_text` , `oa_rollup_1s` , `oa_rollup_1m` , `oa_ingest_status` , and `oa_events` , `oa_executions` , `oa_states` , `oa_context_values` , `oa_relationships` . Report against the views.

Worked example: the packaging line on TimescaleDB, read by Grafana

The packaging line writes into database `meelmo` on `tsdb01` , and the maintenance team wants a Grafana panel of `Packaging/Checkweigher/CaseWeight` against `Packaging/Checkweigher/WeightTarget` .

1. The DBA installs the extension and creates `meelmo_admin`, `meelmo_writer` and `meelmo_reader` as above.
2. On the Desktop, **Data ▸ Historian…**, tick **Keep history in a historian**.
3. **Provider:** `TimescaleDB`. **Connection string:** `Host=tsdb01.plant.local;Database=meelmo;Username=meelmo_admin`; the `meelmo_admin` password.
4. **Test**. It reports the schema is not ready; press **Create schema** and confirm. *Schema created and checked.*
5. **Migration** tab: **Check views**, then **Create views…**, confirm.
6. The DBA runs the `meelmo_writer` and `meelmo_reader` grants.
7. **Store** tab: change the user to `meelmo_writer` and type its password; **Test**.
8. **Ingestion** tab: **Raw retention (days):** `90`, **Build rollups for faster long-range queries** ticked.
9. **Save**.
10. In Grafana, add a PostgreSQL data source for `tsdb01`, database `meelmo`, user `meelmo_reader`, and a time-series panel with:

```
SELECT bucket_utc AS "time", signal_id AS metric, mean_value AS value
FROM meelmo.oa_v_rollup_1m
WHERE signal_id IN ('Packaging/Checkweigher/CaseWeight',
                    'Packaging/Checkweigher/WeightTarget')
  AND $__timeFilter(bucket_utc)
ORDER BY 1;
```

The one-minute tier keeps a panel over several weeks fast. For a panel of the last hour, `oa_v_samples` with `source_time_utc` and `numeric_value` gives every raw sample.

Retention and rollups

Set on the **Ingestion** tab under **Maintenance**: **Raw retention (days):** (`0` keeps everything), **Maintenance interval (min):** (60), **Build rollups for faster long-range queries** (on), **Context store:** (`journal`, or `provider` for the semantic model in these tables).

On TimescaleDB:

- **Retention drops whole chunks.** A `DELETE` over a year of samples writes as much as it removes and leaves the space for a later vacuum; `drop_chunks` unlinks files. The cost is granularity: a chunk covers 7 days, and a chunk is dropped only when it lies entirely before the cutoff. Nothing newer than the cutoff is ever removed, so up to a week more than the retention window may remain on disk.
- **Rollups** go into `oa_rollup_1s` and `oa_rollup_1m` with `ON CONFLICT DO UPDATE`, so an interrupted pass is safe to run again and a late sample is folded into its bucket on the next pass.
- **First pass.** The node waits one maintenance interval before its first pass. **Run maintenance now** in the browser runs one immediately.

Ingestion and the spool

Queue capacity: (200,000), **Batch size:** (5,000), **Flush interval (ms):** (500), **Spool folder:** (`spool`), **Spool limit (GB):** (8) and **When the spool is full:** (`StopAccepting` or `OverwriteOldest`) behave as for every store. A PostgreSQL outage is spooled to disk and replayed; a batch refused as data is kept in `dead-letter.dat` / `dead-letter.log` with the database's reason. Because the inserts deduplicate, a replay that overlaps what already arrived is harmless.

> **Keep the spool at 8 GB, and never below 64 MB.** Back up the spool folder with the historian.

Read alongside and migration from the embedded historian

Read alongside tab, **Add…**: a read-only store — recordings, a Meelmo store kept after a migration, or a database this product did not create — queried with the historian, with an **Id:**, **Name:**, **Priority:** (historian is `0`, lower wins) and **Include this store in queries**.

Migration tab, **Also write to a second store, for a cutover**: **Second store:** `TimescaleDB`, **Second instance id:**, **Connection string:**, **Password:**. The first store stays in charge; the second counts what it misses.

`Meelmo.Migrate` back-fills the older history. The program, its configuration files, every option, the refusal codes and the exit codes are described once, for every store, in *The historian*. What is particular to TimescaleDB:

- **The target file.** `"providerId": "timescale"`, with the settings `connectionString`, `schema`, `copyBatchSize`, `commandTimeoutSeconds` and `allowUnencryptedConnection`, and the password as a sealed token under `secretRefs` with the key `password`. Run the tool on the machine that sealed it.
- **Re-runs are harmless.** TimescaleDB recognises a re-sent sample, so a re-run over windows already copied stores nothing twice, and the profile has no `no-checkpoint` note. Use `--checkpoint` all the same, so an interrupted run resumes where it stopped instead of copying everything again.
- **Into the live store, set a ceiling.** When TimescaleDB is already the store the node acquires into, `--max-rate` keeps the node's own writing ahead of the back-fill.

```
Meelmo.Migrate --source embedded.json --target timescale.json \
             --checkpoint /var/lib/meelmo/migrate/packaging-tsdb.ckpt \
             --include Packaging --dry-run
```

The order: second store on and saved; back-fill with a checkpoint, and with `--to` set to the moment you saved, until the report has no failed windows; confirm the second store lost nothing; make TimescaleDB the historian and turn the second store off.

Reporting views

With the direct-database-access entitlement, **Check views**, **Create views…** and **Remove views…** on the **Migration** tab manage `oa_v_signals`, `oa_v_samples`, `oa_v_rollup_1s`, `oa_v_rollup_1m`, `oa_v_events`, `oa_v_executions` and `oa_v_states` in the historian's schema.

`oa_v_samples` presents the four typed tables as one: `signal_id`, `source_time_utc`, `receive_time_utc`, `quality`, `data_type`, `numeric_value` (booleans as 1 or 0) and `text_value`. The rollup views carry `bucket_utc`, `sample_count`, minimum and maximum with their instants, first, last, `mean_value`, and good and bad counts.

Grant `meelmo_reader` `SELECT` on the views only. A login reading the database directly bypasses Meelmo's per-user signal scoping.

Sizing

Tags	Rate	Raw per day	Raw per year
1,000	1 s	~2.7 GB	~1.0 TB
10,000	1 s	~27 GB	~10 TB
100,000	1 s	~273 GB	~100 TB

Before retention, deadband and any compression you configure in TimescaleDB yourself. For the packaging line's roughly 140 signals at one sample a second, about 0.4 GB a day; with 90 days of retention plus up to one chunk of slack, around 40 GB of sample chunks.

- **Copy batch size** defaults high (50,000) because binary `COPY` is efficient in large batches; lower it only if the server shows memory pressure.
- Size disk for the retention window plus one chunk (7 days) per sample table.
- Autovacuum matters less than on plain PostgreSQL for the sample tables, because retention drops chunks rather than leaving dead rows.

Troubleshooting

Symptom	Cause and remedy
TimescaleDB is *not available here*.	Licence or module; enable `historian-timescale` under **Administration ▸ Modules**.
The provider is *not installed on this machine*.	Re-run the installer with **Database history providers**.
... does not have the timescaledb extension installed ...	Run `CREATE EXTENSION timescaledb;` as a superuser in that database.
Connection failed: ... password authentication failed	Wrong password, or `pg_hba.conf` does not admit the node's address for that user.
Connection failed with an SSL error.	The connection string demands TLS the server does not offer, or the server certificate is not trusted.
'...' is not a valid PostgreSQL schema name.	**Schema** must be a plain identifier.
... exists but is not a hypertable.	The table was not created by **Create schema** with the extension present.
Retention reports it dropped nothing although data is older than the window.	The oldest chunk still contains samples newer than the cutoff. It will go when the whole chunk is older.
Retention fails with a permission error from `drop_chunks`.	The runtime role does not own the sample hypertables; see *Privileges*.
After saving, *the node could not open that store, so this host is still writing to the one it had*.	The setting is saved. Fix what is named and save again, or restart the host.
Degraded (spooling).	PostgreSQL unreachable; samples wait on disk and replay.

QuestDB historian

> **Module: QuestDB historian (`historian-questdb`).** This chapter describes the QuestDB store for the historian. It is present only on installations licensed for that module. Everything about the historian that does not depend on the database — what is recorded, rollups, the history browser, the spool — is in the base chapter *The historian*; this chapter adds what is specific to keeping that history in QuestDB.

What this module adds

QuestDB is a column-oriented time-series database that takes data in over the InfluxDB line protocol and answers SQL over the PostgreSQL wire protocol. This module lets the node's historian write there.

Choose it when the site already runs QuestDB, or wants a fast, single-binary time-series server beside the line without operating PostgreSQL. Choose TimescaleDB instead when the semantic model should live in the same database as the samples, or when a node lease is to be kept there: QuestDB can do neither (see *What it does not hold*). Keep the embedded store when none of that applies.

What the QuestDB store does, in short:

Aspect	Behaviour
Ingestion	Line protocol over HTTP (`/write`), one request per batch, each one answered: committed, refused for something in its lines, or not reached.
Deduplication	Every table is a WAL table with upsert keys on the source time, the signal and the sequence. A re-sent sample replaces the identical row it already wrote, so replays and re-run migrations store nothing twice.
Timestamps	Source and receive time are both `TIMESTAMP_NS` , so every instant round-trips to the 100-nanosecond tick.
Tables	Typed sample tables (numeric, integer, boolean, text) partitioned by day; one-second and one-minute rollups; the signal catalogue. No semantic tables.
Reads	SQL over the PostgreSQL wire. Buckets sit on the same grid as every other store's; first, last and the instants of the extremes are ordered by time and then sequence.
Retention	Drops whole day partitions rather than deleting rows.
Encryption	TLS with the server's certificate verified, on both ports, unless the site turns it off with **Allow an unencrypted connection**. A password or token is never sent unencrypted to another machine.

What it does not hold

- **The semantic model.** Executions, events, states and relationships are rows the product updates in place and two writers must agree on. QuestDB has no in-place update two writers can be ordered by, no transaction spanning statements on its PostgreSQL wire, and no row lock. With QuestDB as the store, **Context store:** must be `journal` (the default). Choosing `provider` is refused when the configuration is checked — *QuestDB cannot hold the semantic model: … Keep the model in the journal* — rather than when the node starts.

- **A node lease.** A redundant pair's lease needs a PostgreSQL both machines can reach. The QuestDB server's PostgreSQL port is not one: the recorder refuses it by name, as a database that can never hold a lease, and stops rather than waiting on it.
- **Reporting views.** The seven `oa_v_*` views are not built on QuestDB. The tables can be queried with SQL — the PostgreSQL wire, the web console, Grafana's QuestDB data source — but their layout is the store's own and may change between versions.

Before you start

1. **The module, licensed and switched on.** The licence must grant the QuestDB provider capability and `historian-questdb` must be enabled under **Administration ▸ Modules**. Otherwise **QuestDB** is listed but marked *not available here*, with the reason.
2. **The provider installed on the node's machine** — the optional **Database history providers** installer component.
3. **QuestDB 9.1 or later**, which has nanosecond timestamps. Tested with 10.0.
4. **Network access from the node** to both ports: HTTP (9000 by default) for samples, and the PostgreSQL wire (8812 by default) for everything else.
5. **Credentials.** One user and password for both ports. On the open source edition that is `pg.user` / `pg.password` for the wire and `http.user` / `http.password` for HTTP, set to the same pair; QuestDB Enterprise can issue a REST token for the HTTP port instead. Change the open source default (`admin` / `quest`) before connecting a plant to it.

Encryption, and the open source edition

QuestDB's open source edition serves neither port over TLS; Enterprise does. So:

- **With QuestDB Enterprise** (or a TLS-terminating proxy in front of each port), leave **Use TLS** on. The server's certificate is checked against the machine's trusted roots, or against **Root certificate** for a private authority, and its name against the host.
- **With the open source edition on this machine** (`localhost` , `127.0.0.1` , `::1`), turn **Use TLS** off and **Allow an unencrypted connection** on. Nothing crosses a network, and a warning is logged at every start.
- **With the open source edition on another machine**, a password or token would cross the network in the clear, and the provider refuses the configuration: *qdb01 is another machine and the connection to it is not encrypted, so its password or token would cross the network in the clear. This installation will not send it.* Put a TLS proxy in front of it, use Enterprise, or run QuestDB beside the node.

Choosing QuestDB as the store

Configure the historian on the Desktop under **Data ▸ Historian…**, or in the browser, with the Web HMI module, under **Administration ▸ Historian**. Both save a configuration revision at the node. Choose **QuestDB** in **Provider:** (Desktop) or **Store** (browser); the line below reads *Stores history in QuestDB: line protocol over HTTP for ingestion, SQL over the PostgreSQL wire for reads, deduplicating WAL tables partitioned by day.* Leave **Instance id:** as `node` for a node's own store.

Connection fields

Field	Required	Default	What to enter
Host	Yes	—	The server's host name or address, without a scheme or a port.
HTTP port	No	`9000`	Where samples are sent as line protocol.

Field	Required	Default	What to enter
PostgreSQL wire port	No	`8812`	Where history is read and maintained with SQL.
User	No	`admin`	The user for both ports.
Password	No	—	Sealed on the node, given to the server at connect time, never displayed again.
HTTP token	No	—	A REST token (Enterprise). Used for the HTTP port instead of the password when set.
Use TLS	No	on	Encrypt both ports and verify the server's certificate.
Root certificate	No	—	A private authority the server's certificate must chain to.
Allow an unencrypted connection	No	off	Only for a server with no TLS, on a segment the site trusts. See above.
Table prefix	No	`oa_`	QuestDB has no schemas; the prefix keeps two installations' tables apart on one server. Letters, digits and underscores.
Rows per request	No	`50000`	Samples per line-protocol request. Clamped to 100–1,000,000.
Command timeout (s)	No	`120`	Seconds a statement, or a request, may run. Clamped to 5–3,600.

Test, Validate and Create schema

All three run at the node.

Action	What it checks or does	Changes the database?
Test	Connects over the PostgreSQL wire, checks the server is QuestDB, and reports whether the schema is ready and whether the HTTP port answers.	No
Validate	Every required table exists; each is a WAL table with upsert keys; none is suspended; the schema version is the one this build expects.	No
Create schema…	Creates the tables. Asks first.	Yes

Samples are not sent before the schema exists. Line protocol creates a table it does not find, with its own guess at the column types and no deduplication, and that table would then be in the way of the real one — so until **Create schema** has run, batches are spooled with *The QuestDB historian's tables (oa_) have not been created*.

Message	Meaning
Tables, write-ahead logs and deduplication are as expected.	**Validate** passed.
The server at … answers the PostgreSQL wire but is not QuestDB	The PostgreSQL wire port points at another server — often a PostgreSQL.
Table oa_… is missing.	Run **Create schema**.
… exists but does not deduplicate (no upsert keys).	The table was created by line protocol, or by hand. Drop it and run **Create schema**.

Message	Meaning
Table oa_… is suspended; QuestDB is not applying its writes.	QuestDB stopped applying that table's log — a full disk is the usual cause. Fix it and run `ALTER TABLE oa_… RESUME WAL;`. Nothing acknowledged is lost meanwhile.

The tables are `oa_schema_version`, `oa_signals`, `oa_samples_numeric`, `oa_samples_integer`, `oa_samples_boolean`, `oa_samples_text`, `oa_rollup_1s` and `oa_rollup_1m` (with the configured prefix in place of `oa_`).

Written, then readable

QuestDB acknowledges a request once it is in the table's write-ahead log, and a query sees it once the log has been applied — usually within milliseconds. The historian waits for its own writes to be applied before it reads, so a chart or an export on the node always includes the last batch. Another program reading the tables directly sees them on QuestDB's schedule.

Retention and rollups

Set on the **Ingestion** tab under **Maintenance**, as for every store. On QuestDB:

- **Retention drops whole day partitions.** The cutoff is taken back to the start of its UTC day, and every partition before that is dropped; nothing newer than the cutoff is ever removed, so up to a day more than the retention window may remain.
- **Rollups** go into `oa_rollup_1s` and `oa_rollup_1m`. Each pass rebuilds from a tier's newest bucket less the late-arrival window, and from anything written behind it since the last pass; the upsert keys make the rebuild a refresh, so an interrupted pass is safe to run again.

Ingestion and the spool

Queue, batches, flush interval and spool behave as for every store. A server that cannot be reached, a credential it refuses (a password rotated under a running node), a port that is not QuestDB's HTTP endpoint, and a server that is busy or starting are all spooled and replayed. A request QuestDB reads and refuses — a value of the wrong type for its column — is kept in `dead-letter.dat` / `dead-letter.log` with QuestDB's reason; later requests of the same batch are still sent.

A sample stamped before 1970-01-01 cannot be stored in QuestDB at all. It is left out of its batch and counted as refused, and the rest of the batch is stored.

Migration

`Meelmo.Migrate` moves history in and out as for every store. The target or source file has `"providerId": "questdb"`, the settings above by key (`host`, `httpPort`, `pgPort`, `username`, `useTls`, `rootCertificate`, `allowUnencryptedConnection`, `tablePrefix`, `rowsPerRequest`, `commandTimeoutSeconds`), and the password (key `password`) and token (key `token`) as sealed tokens under `secretRefs`. Re-runs into QuestDB are harmless; use `--checkpoint` all the same, so an interrupted run resumes where it stopped.

Sizing

Raw volume per sample is about the same as for TimescaleDB before compression. What is particular to QuestDB is that a table's open partition grows its column files in whole pages — 16 MB a column by default — so the eight tables cost a little under a gigabyte before the first sample. That is the server's

`cairo.writer.data.append.page.size`; a small installation can lower it.

Troubleshooting

Symptom	Cause and remedy
QuestDB is *not available here*.	Licence or module; enable `historian-questdb` under **Administration ▸ Modules**.
... is another machine and the connection to it is not encrypted ...	See *Encryption, and the open source edition*.
TLS is switched off for ... and 'allowUnencryptedConnection' is not set.	Turn **Allow an unencrypted connection** on, knowingly, or turn TLS on at the server.
invalid username/password	The wire's user or password is wrong. On the open source edition they are `pg.user` and `pg.password`.
Batches spool with *QuestDB answered 401*.	The HTTP port's credential differs from the wire's; set `http.user` / `http.password` to the same pair, or configure an HTTP token.
QuestDB cannot hold the semantic model ...	Set **Context store:** to `journal`.
Health reads degraded: *its line protocol endpoint does not answer*.	The HTTP port is wrong, firewalled, or HTTP is disabled on the server.
Health reads degraded: *QuestDB has suspended ...*	See the suspended-table message above.

ClickHouse historian

> **Module: ClickHouse historian (`historian-clickhouse`).** This chapter describes the ClickHouse store for the historian. It is present only on installations licensed for that module. Everything about the historian that does not depend on the database — what is recorded, rollups, the history browser, the spool — is in the base chapter *The historian*; this chapter adds what is specific to keeping that history in ClickHouse.

What this module adds

ClickHouse is a column-oriented analytical database. This module lets the node's historian write there, through ClickHouse's own .NET driver over its HTTP interface.

Choose it when the site already runs ClickHouse — often for analytics across many lines or plants — and wants the plant's history in the same place. Choose TimescaleDB instead when the semantic model should live beside the samples, or when a node lease is to be kept there: ClickHouse can do neither (see *What it does not hold*). Keep the embedded store when none of that applies.

What the ClickHouse store does, in short:

Aspect	Behaviour
Ingestion	`INSERT … FORMAT RowBinary` over HTTP(S), one request per sample table, synchronously (`async_insert = 0`): a request is answered once its rows are on the server's disk, and is either stored, refused for something in its rows, or not reached.
Deduplication	Every sample table is a `ReplacingMergeTree` ordered by signal, source time and sequence, and every request carries an insert deduplication token over its rows. A retried request is dropped whole; the same samples sent again in other batches are a second copy on disk until ClickHouse merges the parts, and every read says `FINAL`, so no answer ever counts a sample twice.
Timestamps	Source and receive time are both stored as 64-bit .NET ticks, so every instant round-trips to the 100-nanosecond tick — including a sample from a clock that was never set (year one).
Tables	Typed sample tables (numeric, integer, boolean, text) partitioned by UTC day; one-second and one-minute rollups; the signal catalogue. No semantic tables.
Reads	SQL over HTTP. Buckets sit on the same grid as every other store's; first, last and the instants of the extremes are ordered by time and then sequence.
Retention	Drops whole day partitions rather than deleting rows.
Encryption	HTTPS on 8443 with the server's certificate verified, unless the site turns it off with **Allow an unencrypted connection**. A password is never sent unencrypted to another machine. Redirects are never followed.

What it does not hold

- **The semantic model.** Executions, events, states and relationships are rows the product updates in place and two writers must agree on. ClickHouse has no transactional upsert, no transaction spanning statements and no row lock: an `ALTER … UPDATE` is an asynchronous mutation that rewrites whole parts, and a replacement happens at a merge nobody schedules. With ClickHouse as the

store, **Context store:** must be `journal` (the default). Choosing `provider` is refused when the configuration is checked — *ClickHouse cannot hold the semantic model: … Keep the model in the journal* — rather than when the node starts.

- **A node lease.** A redundant pair's lease needs a PostgreSQL both machines can reach. ClickHouse can serve the PostgreSQL wire protocol, but it is not one: the recorder refuses it by name — *The node lease's connection string names a ClickHouse server* — and stops rather than waiting on it.
- **Reporting views.** The seven `oa_v_*` views are not built on ClickHouse. The tables can be queried with SQL — `clickhouse-client`, the HTTP interface, Grafana's ClickHouse data source — but their layout is the store's own and may change between versions. Query them with `FINAL`, or a replayed sample is counted twice until the parts are merged.

Before you start

1. **The module, licensed and switched on.** The licence must grant the ClickHouse provider capability and `historian-clickhouse` must be enabled under **Administration ▸ Modules**. Otherwise **ClickHouse** is listed but marked *not available here*, with the reason.
2. **The provider installed on the node's machine** — the optional **Database history providers** installer component.
3. **ClickHouse 24.8 or later**, a single server or one replica of a cluster's shard. Tested with 26.3 (LTS).
4. **Network access from the node** to the HTTPS port (8443 by default), or the HTTP port (8123) on this machine.
5. **A user** with the right to insert into and select from the historian's database, and for **Create schema** the right to create the database and its tables. Do not use the passwordless `default` user of a fresh installation.

Encryption

- **On another machine**, serve ClickHouse's HTTPS port (`https_port`, 8443) with a certificate and leave **Use TLS** on. The server's certificate is checked against the machine's trusted roots, or against **Root certificate** for a private authority, and its name against the host.
- **On this machine** (`localhost`, `127.0.0.1`, `::1`), plain HTTP on 8123 is allowed with **Use TLS** off and **Allow an unencrypted connection** on. Nothing crosses a network, and a warning is logged at every start.
- **Plain HTTP to another machine with a password** is refused: *ch01 is another machine and the connection to it is not encrypted, so its password would cross the network in the clear. This installation will not send it.*

Choosing ClickHouse as the store

Configure the historian on the Desktop under **Data ▸ Historian…**, or in the browser, with the Web HMI module, under **Administration ▸ Historian**. Choose **ClickHouse** in **Provider:** (Desktop) or **Store** (browser); the line below reads *Stores history in ClickHouse over its HTTP(S) interface: RowBinary inserts into replacing MergeTree tables partitioned by day, read with FINAL so a replayed sample is never counted twice.* Leave **Instance id:** as `node` for a node's own store.

Connection fields

Field	Required	Default	What to enter
Host	Yes	—	The server's host name or address, without a scheme or a port.

Field	Required	Default	What to enter
HTTP port	No	`8443`	ClickHouse's HTTPS port, or its HTTP port (8123) without TLS.
User	No	`default`	The ClickHouse user.
Password	No	—	Sealed on the node, given to the server at connect time, never displayed again.
Use TLS	No	on	Connect over HTTPS and verify the server's certificate.
Root certificate	No	—	A private authority the server's certificate must chain to.
Allow an unencrypted connection	No	off	Only for a server without TLS, on a segment the site trusts. See above.
Database	No	`meelmo`	The database the tables are in; **Create schema** creates it. ASCII letters, digits and underscores.
Rows per request	No	`50000`	Samples per insert request. Clamped to 100–1,000,000.
Command timeout (s)	No	`120`	Seconds a statement, or a request, may run. Clamped to 5–3,600.

Test, Validate and Create schema

All three run at the node.

Action	What it checks or does	Changes the database?
Test	Connects, reads the server's version, and reports whether the schema is ready and how many active parts it has.	No
Validate	Every required table exists; each is a `ReplacingMergeTree` ordered by the idempotency key; the schema version is the one this build expects.	No
Create schema…	Creates the database and the tables. Asks first.	Yes

Until **Create schema** has run, batches are spooled with *The ClickHouse historian's tables (meelmo.*) have not been created*.

Message	Meaning
Tables, engines and sorting keys are as expected.	**Validate** passed.
Table meelmo.... is missing.	Run **Create schema**.
Table meelmo.... is a MergeTree, not a ReplacingMergeTree.	The table was created by hand or by another tool, and would keep a replayed sample for ever. Drop it and run **Create schema**.
Table meelmo.... is ordered by (...), not (signal_id, source_ticks, source_seq).	The same: a different sorting key would merge different samples, or keep two copies of one.

The tables are `schema_version`, `signals`, `samples_numeric`, `samples_integer`, `samples_boolean`, `samples_text`, `rollup_1s` and `rollup_1m`, in the configured database.

Stored once, eventually; read once, always

ClickHouse deduplicates a `ReplacingMergeTree` when it merges parts — in the background, usually within minutes, but on no promise. A spool replay or a migration run twice therefore leaves a second copy of some samples on disk for a while. Every read the historian makes says `FINAL`, which deduplicates at query time, so charts, exports, statistics, rollups and value lookups are right before the merge and after it. The price was measured on a day of a 10 Hz signal with a tenth of it replayed and not yet merged: a raw day read in about 0.2 s against 0.08 s without `FINAL`, one-minute aggregates in about 0.35 s against 0.15 s; a value lookup finds the nearest key without `FINAL` and reads only that key with it, about 14 ms. A retried request is dropped by its token before it is stored, so the common replay — the same batch again — leaves no second copy at all.

Run `OPTIMIZE TABLE meelmo.samples_numeric FINAL` (and the other sample tables) after a large migration if you want the second copies gone from disk at once.

Retention and rollups

Set on the **Ingestion** tab under **Maintenance**, as for every store. On ClickHouse:

- **Retention drops whole day partitions** (`ALTER TABLE … DROP PARTITION ID`). The cutoff is taken back to the start of its UTC day, and every partition before that is dropped; nothing newer than the cutoff is ever removed, so up to a day more than the retention window may remain. Partitions are numbered by day since 0001-01-01 rather than by date.
- **Rollups** go into `rollup_1s` and `rollup_1m`, rebuilt from deduplicated samples. Each pass starts from a tier's newest bucket less the late-arrival window, and from anything written behind it since the last pass; the tiers replace on signal and bucket, so an interrupted pass is safe to run again.

Ingestion and the spool

Queue, batches, flush interval and spool behave as for every store. A server that cannot be reached, a password it refuses (code 516), a database or table that is not there yet, a server short of memory or with too many parts waiting to merge, and a timeout are all spooled and replayed. Rows ClickHouse reads and refuses — a column of another type than the one the provider writes, rows it cannot parse — are kept in `dead-letter.dat` / `dead-letter.log` with ClickHouse's code and message; later requests of the same batch are still sent. A code nobody listed is retried rather than refused.

Migration

`Meelmo.Migrate` moves history in and out as for every store. The target or source file has `"providerId": "clickhouse"`, the settings above by key (`host`, `port`, `username`, `useTls`, `rootCertificate`, `allowUnencryptedConnection`, `database`, `rowsPerRequest`, `commandTimeoutSeconds`), and the password (key `password`) as a sealed token under `secretRefs`. Re-runs into ClickHouse are harmless (see above); use `--checkpoint` all the same, so an interrupted run resumes where it stopped.

Sizing

ClickHouse compresses every column; the sample tables use delta coding for the timestamps and Gorilla coding for numeric values, so a regularly sampled signal costs a few bytes a sample. Day partitions mean a part per day per table at least: a decade of history is a few thousand partitions per table, which ClickHouse handles but which is the reason the partitioning is by day and not finer. A request may span any number of days — a migration of a slow signal can cover years — at the cost of a part per day it touches.

Troubleshooting

Symptom	Cause and remedy
ClickHouse is *not available here*.	Licence or module; enable `historian-clickhouse` under **Administration ▸ Modules**.
… is another machine and the connection to it is not encrypted …	See *Encryption*.
TLS is switched off for … and 'allowUnencryptedConnection' is not set.	Turn **Allow an unencrypted connection** on, knowingly, or serve HTTPS at the server.
Batches spool with *ClickHouse answered code 516*.	The user or password is wrong.
Batches spool with *code 252* (too many parts).	Inserts are arriving faster than the server merges; raise **Rows per request** or the batch size so fewer, larger requests are sent.
ClickHouse cannot hold the semantic model …	Set **Context store:** to `journal`.
A direct query counts more samples than the historian.	It did not say `FINAL`; see *Stored once, eventually*.

Industrial integration: MQTT, Sparkplug and MCP

> **Module: Industrial integration (`industrial-integration`).** This chapter covers publishing acquired samples to an MQTT broker, publishing as a Sparkplug B edge node, the node's own governed MQTT broker, and the read-only MCP surface through which AI clients query the plant. It is present only on installations licensed for that module.

What this module contains

The module's surfaces are sold together. A licence must grant all three capabilities — MQTT publishing, Sparkplug publishing and the MCP server — for the module to count as licensed; a licence granting two of them does not buy it.

Surface	Direction	What it does
MQTT publication	Outbound	Every acquired sample, as one small JSON message, on a topic derived from the signal id.
Sparkplug B publication	Outbound	The same samples, published as a Sparkplug B edge node into a unified namespace, placed by the asset model.
MCP surface	Inbound, read-only	A Model Context Protocol endpoint on the data-access API, so an AI client can ask about the plant with a service account's token.
MQTT broker	Inbound	A broker on the node holding a governed unified namespace: the node's signals, retained, and what clients publish under access rules and payload contracts.
OPC UA server	Inbound, read-only	The node's signals served to SCADA packages and any OPC UA client: browse, read, subscribe and raw history.
Kafka producer	Outbound	Samples, alarms and events produced to a Kafka cluster, or to Azure Event Hubs through its Kafka endpoint.

Nothing in this module writes to the plant. Publication sends values out; MCP answers questions; the broker carries messages between clients. None of them can change a setpoint in a controller, acknowledge an alarm or alter configuration — a setpoint a client publishes to the broker is a message other clients may read, not a write.

Switching the module on

1. Install the module licence if it arrived separately: **Administration ▸ Modules ▸ Install…** on the Desktop. A module arrives switched off.
2. Select **Industrial integration** in the Modules window and press **Enable…**. Turning a module on is a configuration revision: it is recorded, shows in a diff and can be rolled back.
3. Configure the surface you need, as described below. Each surface is also off until switched on in its own settings.

If the module is switched off, a node that was publishing stops, and the MCP route disappears from the data-access API — it is absent, not refusing, and the host's log says why.

Publishing samples onward

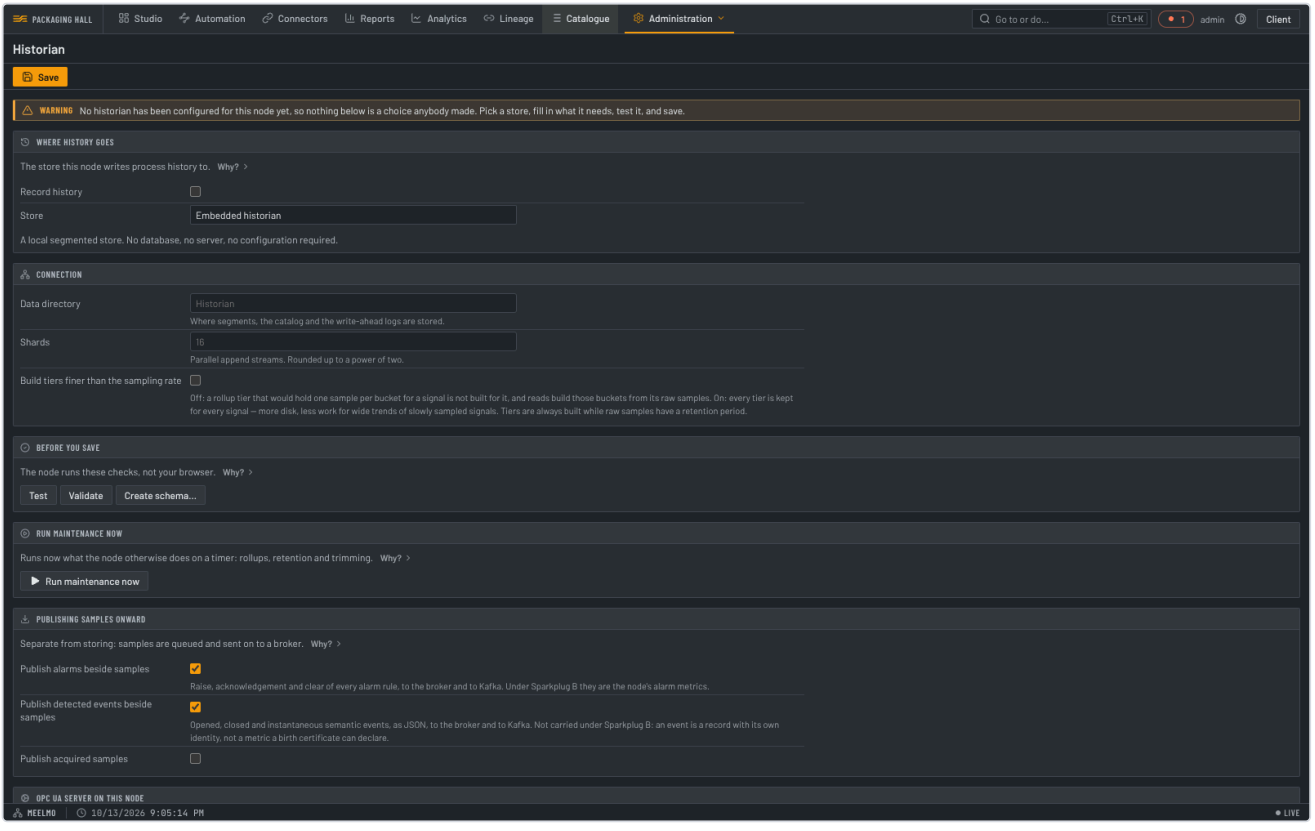
Publishing is a separate job from storing history. A site can store without publishing, or publish to a broker as well as storing. Some rules hold for both transports:

- **Publication needs the historian to be on.** It is one of the historian's sinks. If publication is switched on and the historian is not, nothing is published and the Desktop says so: *Publication is switched on but the historian is not, so nothing is being published.* The historian's store can be any provider, including the embedded one.
- **A slow or absent broker never delays acquisition.** Samples go into a bounded queue and a background pump sends them. When the broker is slow or gone the queue fills, the newest samples are refused and counted, and acquisition is not slowed.
- **Whoever runs the node's acquisition publishes.** On a workstation that hosts its own node, that is the Desktop, while it is running. A plant that must publish around the clock runs the recorder service.
- **Only live samples are published.** Replaying stored history onto a broker is not part of this feature.

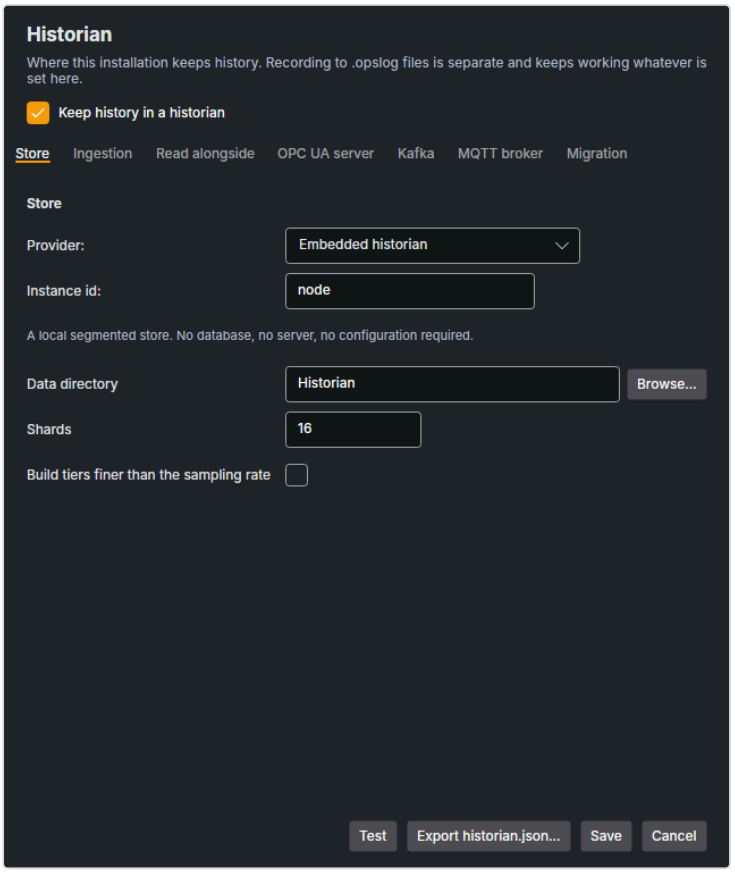
Where it is configured

- On the Desktop: **Data ▸ Historian…**, **Read alongside** tab, section **Publication**.
- In the browser, with the Web HMI module: **Administration ▸ Historian**, panel **Publishing samples onward**.

Both write the same block of the node's configuration, and saving creates a configuration revision.



Administration ▸ Historian in the browser. The last panel, Publishing samples onward, holds the Publish acquired samples switch; ticking it reveals the transport, broker and topic settings



The Desktop's Historian settings window. Publication is on its Read alongside tab, beside the read-only sources; the Store tab is shown here

Settings common to both transports

Setting (Desktop / browser)	Default	Meaning
Publish acquired samples	Off	Sending plant data to somebody else's infrastructure is never a default.
Transport: / **Transport**	MQTT	**MQTT** or **Sparkplug B**. They are different contracts, not variations of one (see below).
Broker host: / **Broker host**	`localhost`	The broker's name or address.
Broker port: / **Broker port**	`1883`	Use the broker's TLS port (usually `8883`) with **Connect over TLS**.
Client id: / **Client id**	`meelmo-publisher`	How this installation identifies itself to the broker. Two publishers sharing one id disconnect each other.
Username: / **Username**	—	If the broker requires one.
Password: / **Password**	—	Sealed on the node and never shown again. An empty box keeps the stored one. In the browser, **Forget the stored password** removes it.
Connect over TLS	Off	Encrypts the connection to the broker.
Accept a certificate that does not validate (browser)	Off	Only shown with TLS. The page warns that a publisher trusting anything on the far end is publishing in the clear with extra steps. Install the broker's CA instead.
Publish only samples the driver marked good	Off	Leaves out samples with a bad or uncertain quality code.

Setting (Desktop / browser)	Default	Meaning
Queue capacity (browser)	`50000`	Samples that may be waiting to go out. When full, the newest are refused and counted.

The client connects with MQTT 3.1.1, which every plant broker understands, and retries with MQTT 5.0 if the broker refuses 3.1.1. A lost connection is re-tried with a back-off of up to 30 seconds.

The MQTT transport

Choose **MQTT** for one JSON message per sample, on topics you choose the root of.

Setting	Default	Meaning
Topic prefix: / **Topic prefix**	`meelmo`	Every topic begins with this, so a broker shared with other systems has a namespace to route on. Empty publishes at the root. May not contain `+` or `#`.
Quality of service (browser)	`0 – at most once`	Also `1 – at least once` and `2 – exactly once`.
Ask the broker to keep the last value per topic	Off	The retain flag. A retained sample outlives the process that published it and is handed to a new subscriber long after it stopped being true, so leave it off unless every consumer understands that.

Topics

The topic is the prefix followed by the signal's stable id, unchanged:

```
<prefix>/<signal id>
meelmo/Packaging/Checkweigher/CaseWeight
```

The product does not invent structure from names. Characters an MQTT topic cannot carry literally — the wildcards `+` and `#`, and the null character — are replaced by `_` so that every signal stays addressable and no signal lands on a wildcard.

Payload

One UTF-8 JSON object per sample, with four short fields:

Field	Meaning
`t`	Source timestamp, ISO-8601 with offset (UTC).
`v`	The value: a number, `true` / `false`, a string, or `null` for a numeric value that is not a finite number.
`q`	The quality code the driver reported, as a number. `0` is good. It is not translated into a word, because the codes belong to your protocol.
`s`	The source sequence number, when the driver supplies one.
`r`	Why the value is not current, when there is a reason to give. Today only on the last message a signal is sent when an activation removes it from the configuration: `"v":null`, `"q":2` and `"r":"Removed from the configuration in revision N."`. Sent even with **Publish only samples the driver marked good**, so a topic does not go on holding the last good reading of a signal the node no longer has.

```
{"t":"2026-09-29T08:14:03.2500000+00:00","v":12.47,"q":0}
```

Sparkplug B has no quality field. A value that is not good carries a `Quality` property — OPC DA codes, `0` bad and `64` uncertain, the convention Ignition and Meelmo's own Sparkplug driver read — and a signal an activation removed is sent once as a null metric with `Quality` 0, then left out of the next birth certificate.

Worked example: packaging line to the site broker

The site runs a broker at `mqtt.plant.local` on port 8883 with TLS, and a dashboard team subscribes to `site1/packaging/#`.

1. Open **Data ▸ Historian…** on the Desktop (the historian must already be on, with any store).
2. On **Read alongside**, under **Publication**, tick **Publish acquired samples**.
3. **Transport:** `MQTT`. **Broker host:** `mqtt.plant.local`. **Broker port:** `8883`. Tick **Connect over TLS**.
4. **Client id:** `meelmo-packaging-line`. **Username:** `meelmo`, and the password.
5. **Topic prefix:** `site1/packaging`. Leave the retain box unticked.
6. Tick **Publish only samples the driver marked good** if the dashboard should never see a bad reading.
7. **Save**.

A subscriber to `site1/packaging/#` now receives, for example:

```
site1/packaging/Packaging/Checkweigher/CaseWeight   {"t":"…","v":12.47,"q":0}
site1/packaging/Packaging/Loader/Rate               {"t":"…","v":29.4,"q":0}
site1/packaging/Packaging/Erector/Running           {"t":"…","v":true,"q":0}
```

In the browser, the same settings are in **Administration ▸ Historian ▸ Publishing samples onward**, where **Quality of service** and **Queue capacity** are also offered.

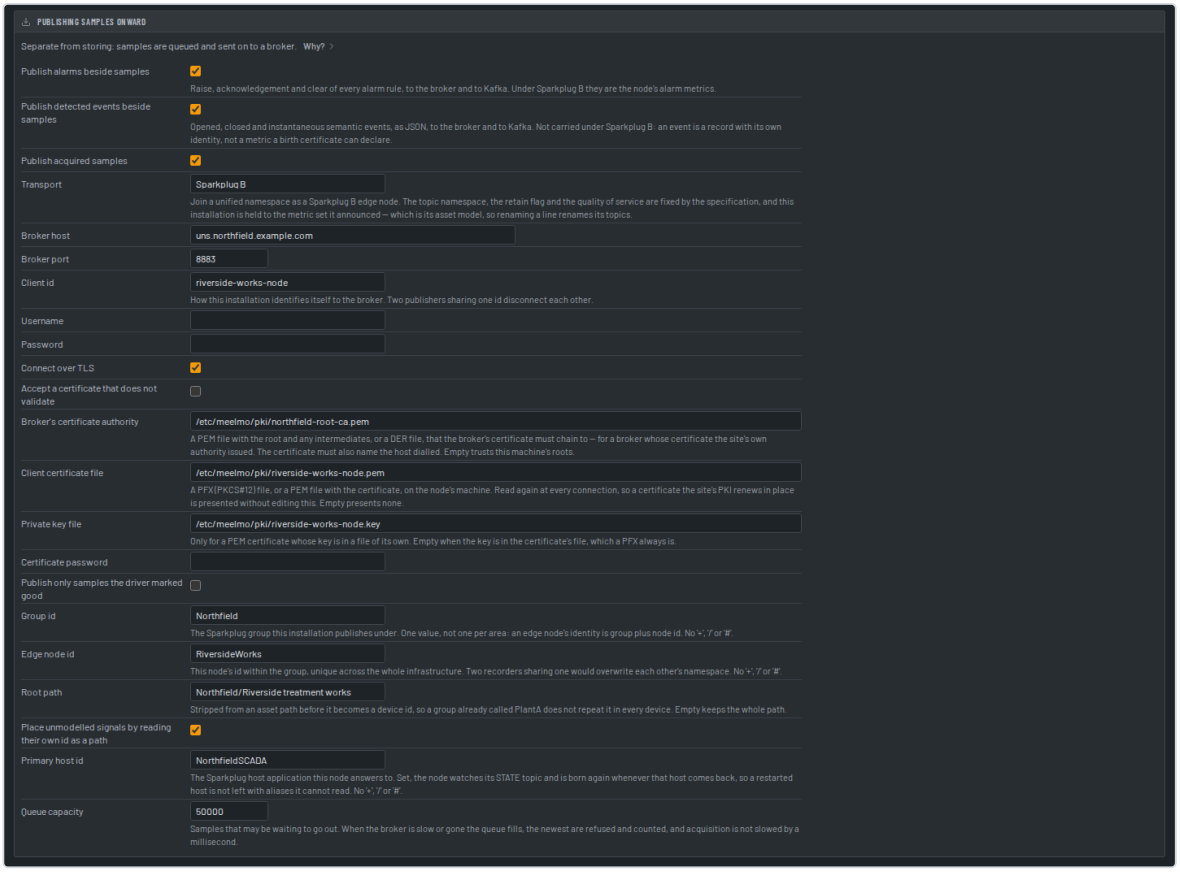
The Sparkplug B transport

Choose **Sparkplug B** to join a unified namespace as an edge node. Under Sparkplug the topic namespace, the retain flag and the quality of service of each message type are fixed by the specification, so **Topic prefix**, **Quality of service** and the retain box disappear from the form. In their place:

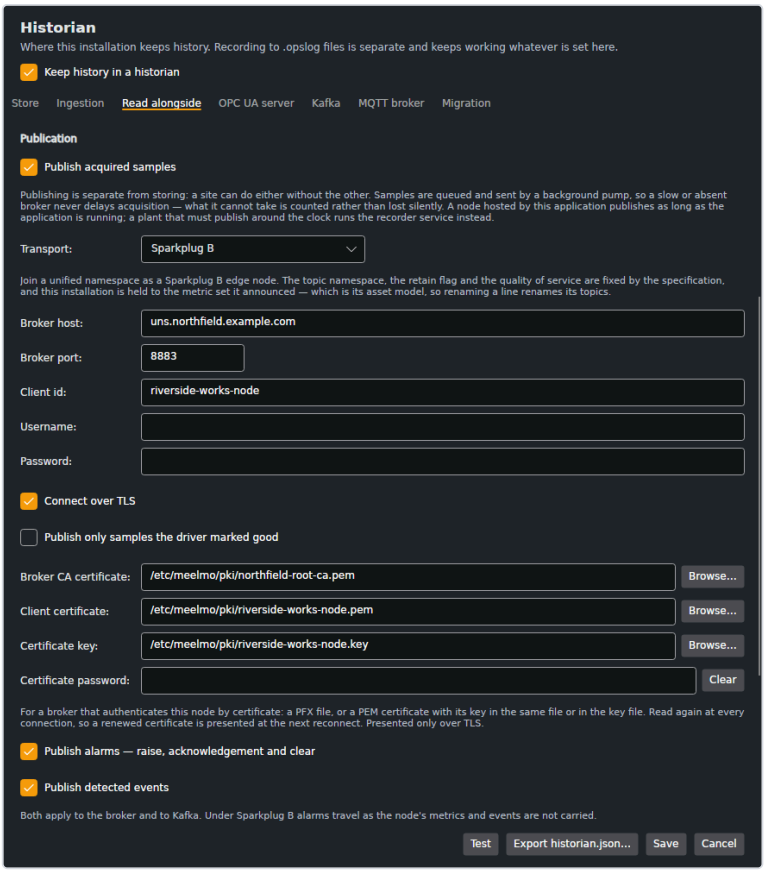
Setting	Default	Meaning
Group id: / **Group id**	`Meelmo`	The Sparkplug group this installation publishes under. One value per installation, not one per area. No `+`, `/` or `#`.
Edge node id: / **Edge node id**	The machine name	This node's id within the group, unique across the whole infrastructure. Two nodes sharing one would overwrite each other's namespace. No `+`, `/` or `#`.
Root path: / **Root path**	empty	Stripped from an asset path before it becomes a device id, so a group already called after the plant does not repeat it in every device. Must not begin or end with `/`.
Place unmodelled signals by reading their own id as a path	On	Signals the asset model says nothing about are placed by reading their id as `Area/…/Machine/Tag`. Such placements are reported as inferred, never as modelled.
Primary host id: / **Primary host id**	empty	The Sparkplug host application this node answers to. The node watches that host's STATE topic and is born again whenever it comes back, so a restarted host learns every alias afresh.

A broker on the site's PKI

An enterprise broker that admits clients by certificate is answered with **Client certificate file** (a PFX, or a PEM certificate with its key in **Private key file**), and one whose own certificate the site's authority issued with **Broker's certificate authority**, so that broker is trusted for this connection alone rather than machine-wide. Both appear once **Connect over TLS** is ticked; they are paths on the node's machine, read again at every connection, so a certificate the site's PKI renews in place is presented at the next reconnect. On the Desktop they are **Broker CA certificate:**, **Client certificate:** and **Certificate key:**.



Publishing samples onward on the reference works: Sparkplug B to uns.northfield.example.com on port 8883 over TLS, the broker's certificate authority, the node's client certificate and private key from /etc/meelmo/pki, group Northfield, edge node RiversideWorks, root path Northfield/Riverside treatment works and primary host NorthfieldSCADA, with alarms and detected events published beside the samples



The same publication on the Desktop: Data ▸ Historian..., Read alongside tab, scrolled to Publication — Sparkplug B over TLS with the broker CA certificate, client certificate and certificate key, and alarms and detected events published

What goes on the wire

Topics follow the specification:

```
spBv1.0/<group id>/<message type>/<edge node id>[/<device id>]
```

Message	When
`NBIRTH`, `DBIRTH`	Immediately after every connection, and again on request. A birth lists every metric with its name, type and alias.
`NDATA`, `DDATA`	Values, carrying aliases. At most 250 metrics per message; a larger poll is split.
`NDEATH`	Registered with the broker as the MQTT Will before connecting, so the broker announces the node's death even if the process is killed.
`NCMD`	The node subscribes to its own command topic and honours `Node Control/Rebirth`, so a host application that restarted can get the alias table back.

The namespace is the asset model. A signal attached to an asset in the semantic model publishes on that asset's device; its metric name is the signal's name within the asset. The asset path becomes one topic level by replacing `/` with `:`. Rename a line in the asset model, and its topics are renamed with it.

A signal that appears after the birth certificate (a new tag, a new virtual signal) is dropped and counted until the next birth; the node re-announces itself at most once every 30 seconds so that a stream of unknown signals cannot flood the broker with births. The first birth after start-up may be incomplete while the catalogue fills on the first poll; the rebirth corrects it.

Worked example: the packaging line as a Sparkplug edge node

The plant's unified namespace uses group `PlantA`. The packaging line's node is to appear as edge node `PACK-LINE-1`.

1. **Data ▸ Historian… ▸ Read alongside ▸ Publication**, tick **Publish acquired samples**.
2. **Transport:** `Sparkplug B`. **Broker host:** `mqtt.plant.local`, **Broker port:** `8883`, **Connect over TLS** ticked, **Client id:** `meelmo-pack-line-1`.
3. **Group id:** `PlantA`. **Edge node id:** `PACK-LINE-1`.
4. **Root path:** `Packaging`, so devices are named after the machine rather than repeating the line.
5. Leave **Place unmodelled signals by reading their own id as a path** ticked while the asset model is being completed.
6. **Save**.

With no asset model yet, `Packaging/Checkweigher/CaseWeight` is placed from its own id: the device is `Checkweigher` (the path `Packaging/Checkweigher` with the root path removed) and the metric is `CaseWeight`. The node publishes:

```
spBv1.0/PlantA/NBIRTH/PACK-LINE-1
spBv1.0/PlantA/DBIRTH/PACK-LINE-1/Checkweigher      metrics CaseWeight, WeightTarget, Running, …
spBv1.0/PlantA/DDATA/PACK-LINE-1/Checkweigher       CaseWeight = 12.47
spBv1.0/PlantA/DDATA/PACK-LINE-1/Erector            MagazineLevel = 63, Running = true
```

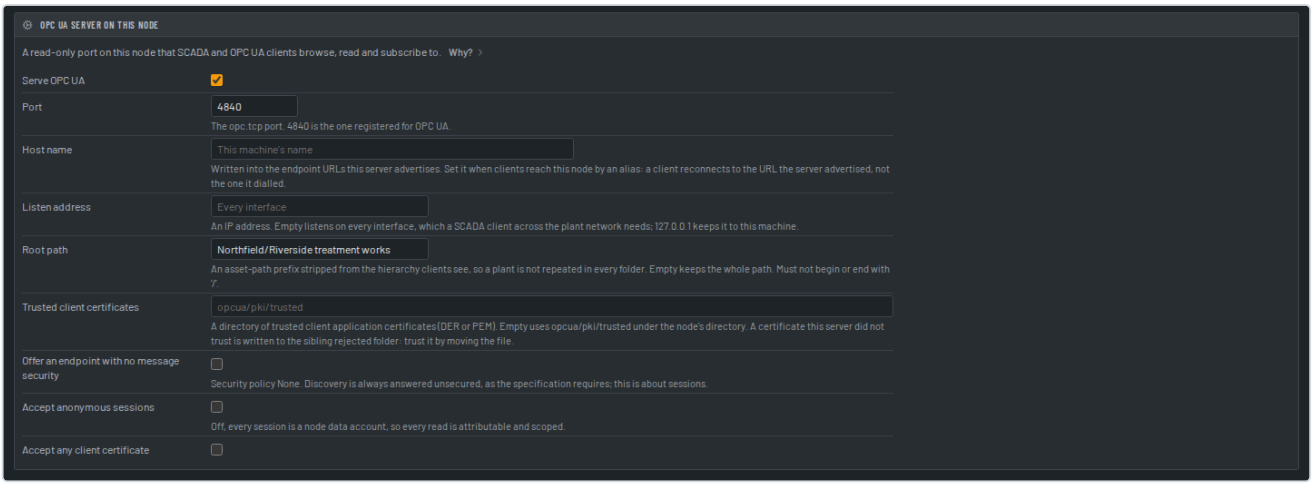
Had **Root path** been left empty, the device would be `Packaging:Checkweigher`.

The OPC UA server

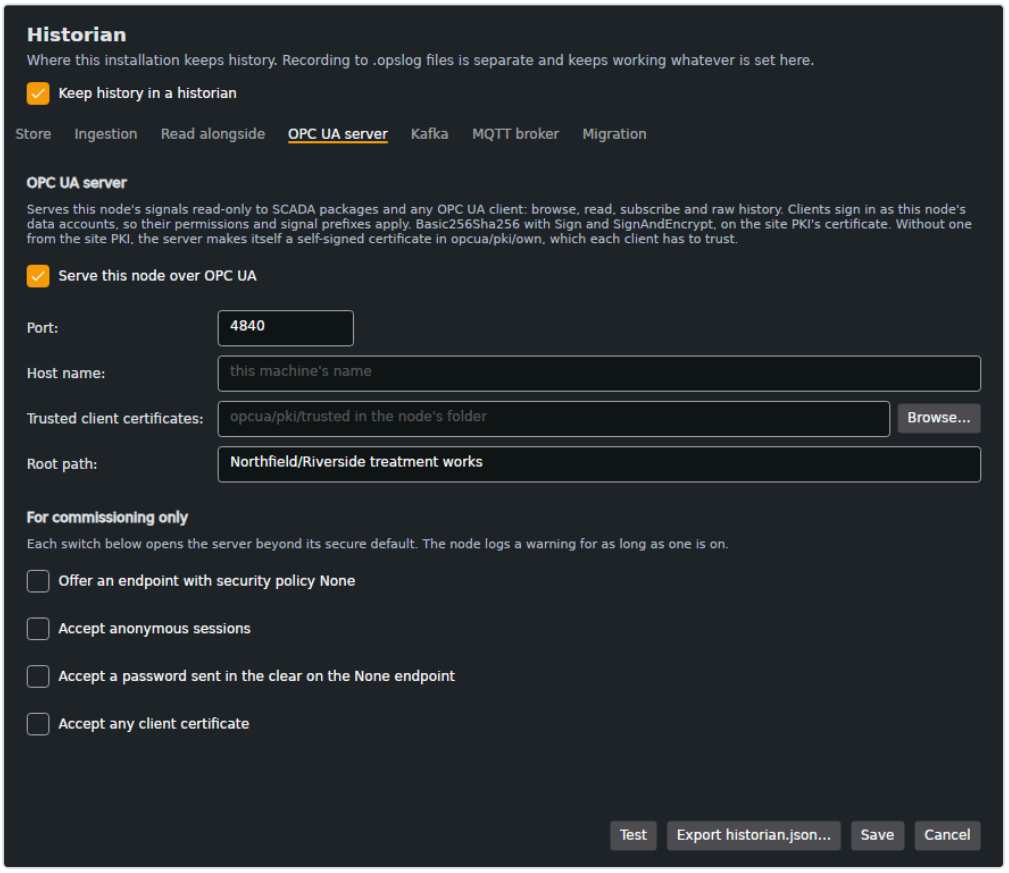
The node can serve its own signals over OPC UA, read-only: a SCADA package, Kepware or any OPC UA client browses them under the asset hierarchy, reads them with source and server timestamps, subscribes with a deadband and reads raw history in bounded pages. Clients sign in as the node's data accounts, so each account's permissions and signal prefixes apply; writes, method calls and history updates are refused. It is set in **Administration ▸ Historian**, panel **OPC UA server on this node**, and on the Desktop in **Data ▸ Historian…**, tab **OPC UA server**, and is off by default. Its secured endpoints (Basic256Sha256, Sign and SignAndEncrypt) use the certificate the site PKI issued the node; every switch under the commissioning heading opens the server beyond that default, and each one, while on, is said in a warning under the panel.

A node the site PKI has issued no certificate — one that has not adopted the managed PKI — still offers both secured endpoints: on first start the server makes itself a self-signed application certificate, as every OPC UA product does, and keeps it in `opcua/pki/own` under the node's directory (`meelmo-node.der`, its key sealed beside it). It trusts nobody by it. A client trusts the server by copying that file into its own trusted certificates, as the server trusts a client by moving its certificate into `opcua/pki/trusted`; the node logs the certificate's thumbprint when it starts serving. The same certificate is presented after every restart, and once the site PKI issues the node one, that one is presented from the next connection on. Should the server offer no endpoint at all — no certificate it can present and security policy None off — the node's publication status says it is not serving, and why.

One Meelmo node reading another's server. The OPC UA driver's client (see *OPC UA message security* in the *Acquisition* chapter) trusts this server's certificate without a file being copied when the site PKI issued it; a server on its own self-signed certificate is refused the first time, its certificate lands in the reading node's `opcua/client-pki/rejected`, and moving it to `opcua/client-pki/trusted` trusts it. This server, in turn, trusts the client by its certificate in `opcua/pki/trusted` — the client's is in this server's `opcua/pki/rejected` after its first attempt. Both ends use one implementation of the security policies, shared between the server and the client.



OPC UA server on this node, switched on for the reference works: port 4840, the host name and listen address left to this machine and every interface, root path Northfield/Riverside treatment works so the plant is not repeated in every folder, the trusted client certificates in opcua/pki/trusted, and none of the commissioning switches on



The same server on the Desktop: Data ▸ Historian..., OPC UA server tab, with the four For commissioning only switches off

Server fields

Field (browser / Desktop)	Default	Meaning
Serve OPC UA / **Serve this node over OPC UA**	Off	Turns the server on. In the browser the other fields appear only when it is ticked.

Field (browser / Desktop)	Default	Meaning
Port / **Port:**	`4840`	The `opc.tcp` port, from 1 to 65535. 4840 is the port registered for OPC UA.
Host name / **Host name:**	This machine's name	Written into the endpoint URLs the server advertises, `opc.tcp://<host name>:<port>`. Set it when clients reach the node by an alias: a client reconnects to the URL the server advertised, not to the one it dialled.
Listen address (browser only)	Every interface	An IP address. Empty listens on every interface, which a SCADA client across the plant network needs; `127.0.0.1` keeps the server to this machine.
Root path / **Root path:**	Empty	An asset-path prefix removed from the hierarchy clients browse, so the plant is not repeated in every folder. Empty keeps the whole path. Must not begin or end with `/`.
Trusted client certificates / **Trusted client certificates:**	`opcua/pki/trusted` under the node's directory	A folder of trusted client application certificates (DER or PEM). A certificate the server did not trust is written to the `rejected` folder beside it; trust it by moving the file into this folder.

For commissioning only

Each of these switches opens the server beyond its secure default. All four are off by default, and while any one is on the node logs a warning and the browser shows it under the panel.

Switch (browser / Desktop)	What it allows	Warning shown while it is on
Offer an endpoint with no message security / **Offer an endpoint with security policy None**	Sessions over security policy None. Discovery is always answered unsecured, as the specification requires; this switch is about sessions.	*A session over the unsecured endpoint is readable and forgeable by anybody on the wire.*
Accept a cleartext password on that endpoint / **Accept a password sent in the clear on the None endpoint**	A data account's password sent unencrypted on the None endpoint, for a client that cannot encrypt it (this product's own built-in client among them). In the browser it appears only while the None endpoint is offered, and clearing that endpoint clears it too.	*The password of a node data account crosses the network in the clear whenever a client uses it.*
Accept anonymous sessions	Sessions with no account. Off, every session is a node data account, so every read is attributable and scoped.	*An anonymous session may read every signal this node has, and nobody's name is on the read.*
Accept any client certificate	Every client application, whatever its certificate.	In red, *Every client application is trusted*: any program that can reach the port is admitted. The node also logs a warning at start-up and on every connection.

A client signs in with a user name and password: the user name is a node data account's id and the password is its token. That account's permissions and signal prefixes decide what the client can browse and read.

Worked example: the packaging line for the site SCADA

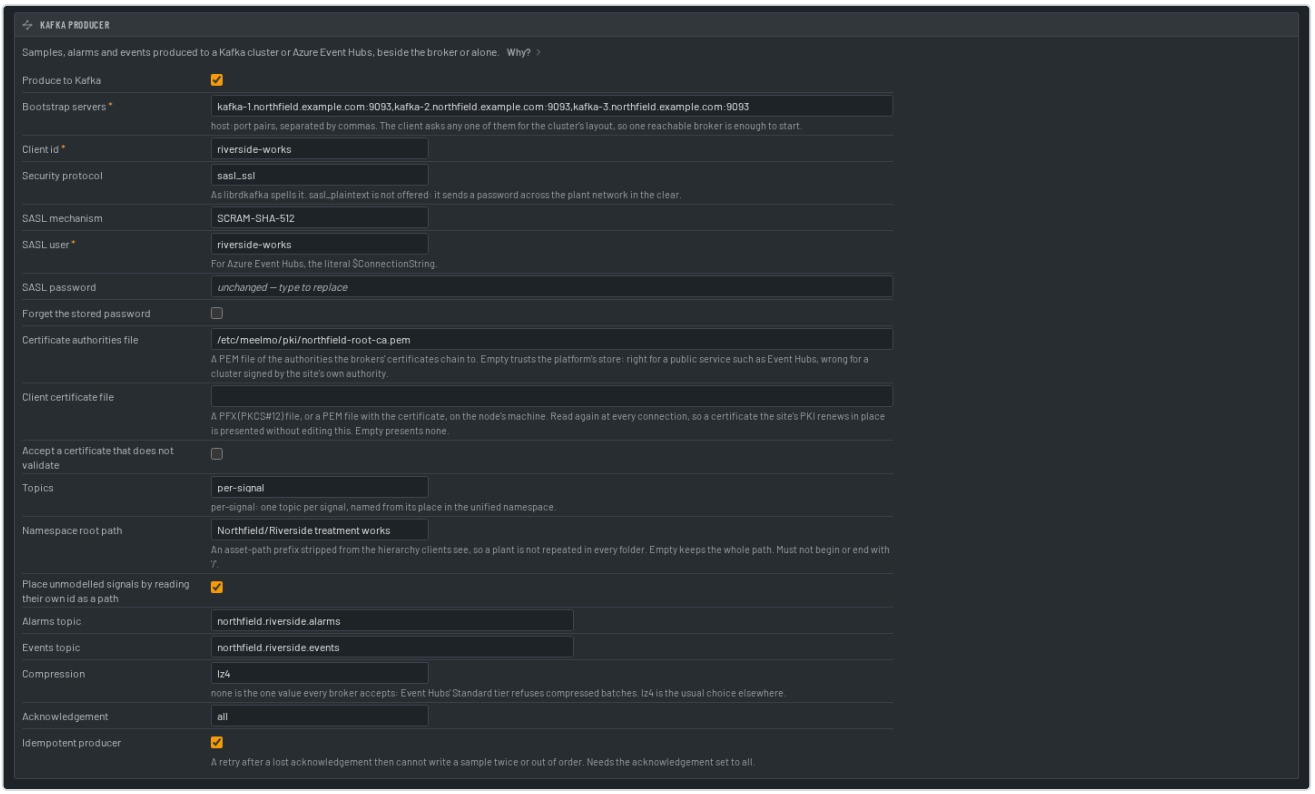
The site's SCADA package should read the packaging line from the node, which the plant network knows as `pack-node.plant.local`.

1. Create a data account for the SCADA package under **Administration ▸ Service Accounts…**, for example `scada-packaging`, with a read-only role and its readable signals limited to the packaging line. Copy its token when it is shown.
2. Open **Administration ▸ Historian**, panel **OPC UA server on this node** (on the Desktop: **Data ▸ Historian…**, tab **OPC UA server**). Tick **Serve OPC UA**.
3. Leave **Port** at `4840`. Set **Host name** to `pack-node.plant.local`, so the endpoint the server advertises is `opc.tcp://pack-node.plant.local:4840`. Leave **Listen address** empty, so the SCADA server on another machine can reach it.
4. Leave **Root path** and **Trusted client certificates** empty, and every switch under the commissioning heading off. Save.
5. In the SCADA package, add the server `opc.tcp://pack-node.plant.local:4840` with security policy Basic256Sha256, mode SignAndEncrypt, user name `scada-packaging` and the token as password. If the node has no certificate from the site PKI, copy `opcua/pki/own/meelmo-node.der` from the node's directory into the client's trusted certificates first.
6. The first connection is refused, because the server does not yet trust the client's certificate: the client's certificate appears in `opcua/pki/rejected`. Move it into `opcua/pki/trusted` and connect again.

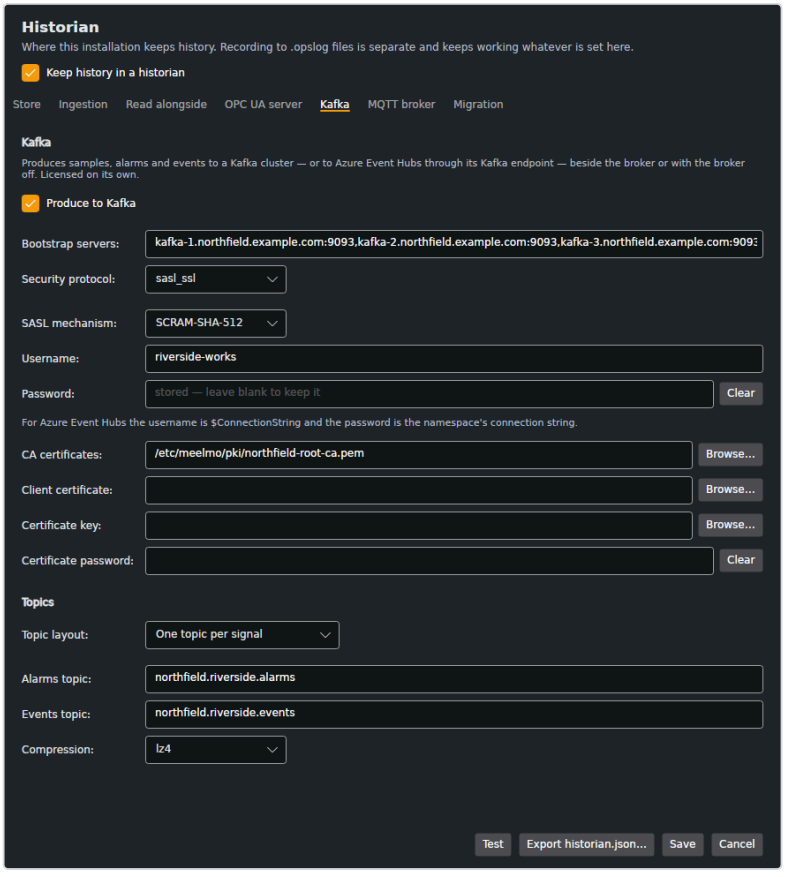
The SCADA package now browses the packaging signals, reads them with source and server timestamps, subscribes to them, and reads their raw history. A write from it is refused.

Kafka

A Kafka producer beside the broker, or alone: samples, alarms and events produced to a Kafka cluster, or to Azure Event Hubs through its Kafka endpoint. It has its own switch and its own licence, and its settings are spelled as librdkafka spells them — `sasl_ssl`, `SCRAM-SHA-512`, `lz4` — so a value copied from the cluster's documentation needs no translating. It is set in **Administration ▸ Historian**, panel **Kafka producer**, and on the Desktop in **Data ▸ Historian…**, tab **Kafka**. Not in the Community download.



The Kafka producer of the reference works: three bootstrap servers on port 9093, client id riverside-works, sasl_ssl with SCRAM-SHA-512 and a stored password, the site's root authority, one topic per signal under Northfield/Riverside treatment works, alarms and events topics northfield.riverside.alarms and northfield.riverside.events, lz4 compression, acknowledgement all and an idempotent producer



The same producer on the Desktop: Data ▸ Historian..., Kafka tab

Producer fields

The browser's **Kafka producer** panel offers every field below; the Desktop's **Kafka** tab offers the ones with a Desktop label and keeps the others as they are. Values are spelled as librdkafka spells them.

Field (browser / Desktop)	Default	Meaning
Produce to Kafka / **Produce to Kafka**	Off	Turns the producer on, beside the MQTT or Sparkplug publication or with it off.
Bootstrap servers / **Bootstrap servers:**	`localhost:9092`	Required. `host:port` pairs separated by commas. The producer asks any one of them for the cluster's layout, so one reachable broker is enough to start.
Client id (browser only)	`meelmo-publisher`	Required. How this producer identifies itself to the cluster.
Security protocol / **Security protocol:**	`plaintext`	`plaintext`, `ssl` or `sasl_ssl`. `sasl_plaintext` is not offered: it sends a password across the plant network in the clear.
SASL mechanism / **SASL mechanism:**	`PLAIN`	With `sasl_ssl` only. `PLAIN`, `SCRAM-SHA-256` or `SCRAM-SHA-512`.
SASL user / **Username:**	—	Required with `sasl_ssl`. For Azure Event Hubs, the literal `$ConnectionString`.
SASL password / **Password:**	—	With `sasl_ssl`. Sealed on the node; an empty box keeps the stored one. In the browser, **Forget the stored password** removes it. For Azure Event Hubs, the namespace's connection string.
Certificate authorities file / **CA certificates:**	The platform's trust store	With `ssl` or `sasl_ssl`. A PEM file of the authorities the brokers' certificates chain to. Empty is right for a public service such as Event Hubs and wrong for a cluster signed by the site's own authority.
Client certificate file / **Client certificate:**	None	With `ssl` or `sasl_ssl`. A PFX file, or a PEM file with the certificate, on the node's machine, read again at every connection.
Private key file / **Certificate key:**	None	Only for a PEM certificate whose key is in a file of its own. A key without a certificate is refused.
Certificate password / **Certificate password:**	—	The PFX password or the PEM key's passphrase. Sealed on the node. In the browser, **Forget the stored certificate password** removes it.
Accept a certificate that does not validate (browser only)	Off	With `ssl` or `sasl_ssl`. Trusts whatever is on the far end; the page warns while it is on. Give the cluster's authority instead.
Topics / **Topic layout:**	`single` (**One topic for every sample**)	`single`: every sample on one topic, keyed by its signal id. `per-signal` (**One topic per signal**): one topic per signal, named from its place in the unified namespace (see below).
Samples topic / **Samples topic:**	`meelmo.samples`	With `single` only. The topic every sample goes to.
Namespace root path (browser only)	Empty	With `per-signal` only. An asset-path prefix removed before the topic is named. Must not begin or end with `/`.

Field (browser / Desktop)	Default	Meaning
Place unmodelled signals by reading their own id as a path (browser only)	On	With `per-signal` only. Signals the asset model says nothing about are placed by reading their id as a path, as Sparkplug does.
Alarms topic / **Alarms topic:**	`meelmo.alarms`	Alarms, keyed by the alarm rule's id.
Events topic / **Events topic:**	`meelmo.events`	Detected events, keyed by the asset's id (or the event's own id when it has no asset).
Compression / **Compression:**	`none`	`none`, `gzip`, `snappy`, `lz4` or `zstd`. `none` is the one value every broker accepts: Event Hubs' Standard tier refuses compressed batches. `lz4` is the usual choice elsewhere.
Acknowledgement (browser only)	`all`	`all`, `leader` or `none`: how many of the cluster's replicas must have a message before it counts as delivered.
Idempotent producer (browser only)	On	A retry after a lost acknowledgement then cannot write a sample twice or out of order. Needs **Acknowledgement** `all`; any other value with it on is refused when you save.

A topic name you type may hold only letters, digits, `.`, `_` and `-`, at most 249 characters, and may not be `.` or `..`.

Each sample is one message, with the same JSON as the MQTT transport (`t`, `v`, `q` and, when the driver gives one, `s`). Under `per-signal`, the topic is the signal's asset path with the root path removed and the path's `/` turned into `.`; a character Kafka does not allow becomes `_`, and a path longer than 249 characters is cut and given a hash of the whole, so every signal gets a topic of its own. The producer shares the publication queue with the MQTT or Sparkplug publication: when the cluster does not keep up, the newest samples are refused and counted, and acquisition is not slowed.

Worked example: the packaging line to the enterprise's Kafka cluster

The enterprise's data platform reads plant data from a Kafka cluster with three brokers, `kafka-1.corp.example`, `kafka-2.corp.example` and `kafka-3.corp.example`, on port 9093, signed by the company's own certificate authority and secured with SCRAM.

1. Open **Administration ▸ Historian**, panel **Kafka producer**, and tick **Produce to Kafka**.
2. **Bootstrap servers:** `kafka-1.corp.example:9093,kafka-2.corp.example:9093,kafka-3.corp.example:9093`. **Client id:** `packaging-line-1`.
3. **Security protocol:** `sasl_ssl`. **SASL mechanism:** `SCRAM-SHA-512`. **SASL user** and **SASL password** as the platform team issued them.
4. **Certificate authorities file:** the company authority's PEM file on the node, for example `C:\ProgramData\Meelmo\pki\corp-root.pem`. Leave **Accept a certificate that does not validate** cleared.
5. **Topics:** `single`. **Samples topic:** `packaging.samples`. **Alarms topic:** `packaging.alarms`. **Events topic:** `packaging.events`.
6. **Compression:** `lz4`. Leave **Acknowledgement** at `all` and **Idempotent producer** ticked. Save.

A consumer of `packaging.samples` now receives one message per sample, with the key `Packaging/Checkweigher/CaseWeight` and the value `{"t":"…","v":12.47,"q":0}`. Had you chosen `per-signal` with **Namespace root path** `Packaging` and no asset model yet, the same samples would go to the topic `Checkweigher.CaseWeight`.

The MCP surface for AI clients

The Model Context Protocol (MCP) surface lets an AI client — a desktop assistant, an agent your integrators run — ask about the plant in the plant's own terms: which signals exist, what happened during a run, what relates to what. It is **read-only**: no tool on it asks for a write capability.

It is served by the node's data-access API at:

```
POST https://<host>:7443/api/v1/mcp      JSON-RPC 2.0
```

Port 7443 is the data-access API's default. The route is part of the data-access API, not of the web HMI.

Three gates, all required

The route exists only when all three hold. Otherwise it is absent rather than refusing, and the host logs the reason.

Gate	How
The switch	On the Desktop, **Recording ▸ Recorder Service…**, section **Data-access API (REST, WebSocket and gRPC)**: tick **Serve the data-access API** and **Serve the MCP surface, for AI assistants that query this plant's history**. For a standalone data-access server, set the environment variable `MEELMO_MCP=true`. Off unless asked for.
The licence	The MCP server capability, part of this module's licence.
The module	`industrial-integration` installed and enabled on the node.

The account a client uses

An AI client authenticates with a **service account's bearer token**. Service accounts are created on the Desktop under **Administration ▸ Service Accounts…**.

- **The account must hold the `Mcp` permission.** No role preset ticks it. Tick it only for the account an assistant will use.
- **Its other permissions still apply, tool by tool.** A token without `ReadCatalog` gets a refusal from `catalog_search` naming that capability, not an empty list.
- **Its signal scope still applies.** An account limited to `Packaging/Checkweigher` sees only those signals, and only the assets, runs, events and states entirely within its scope.

Beside the permission, the same window's **AI assistants (MCP)** section sets how much of the plant a model using the account may be handed. Leave a field blank for the default:

Field	Default	What happens at the limit
Samples per tool call	20,000	A history read is shared out across its signals and marked truncated. Narrowed, never refused.
KiB of answers per minute	4,096	An answer that would exceed the minute is refused with a time to retry.
May read raw samples (otherwise the server returns buckets)	On	Off: a read asking for raw samples is refused, and other reads get buckets of at least one second.

The tools

There are 33 tools, each with its own permission check:

Group	Tools
Catalogue	`catalog_search`, `signal_availability`
History	`history_read`, `history_statistics`, `history_evidence`, `history_value_at`
Live	`realtime_current`, `alarms_active`, `alarms_performance`, `advisories`
Semantic	`semantic_assets`, `semantic_types`, `semantic_executions`, `semantic_events`, `semantic_states`, `semantic_context`, `semantic_relationships`
Datasets	`dataset_definitions`, `dataset_run`
Analysis	`run_compare`, `analysis_correlations`, `analysis_occurrences`, `analysis_contribution`, `analysis_shapes`, `analysis_line_time`
Rules	`analysis_fingerprint`, `analysis_advisory_replay`
Models	`model_candidates`, `model_fit`, `model_estimate`, `model_watch_tune`, `model_watch_replay`
The record itself	`historian_status`

The analysis and models tools are questions the product already answers, not raw material: a client asks *what moved around this moment* instead of pulling a week of samples and doing arithmetic in its own context. The models tools answer only where the Models module is licensed. Each answer carries its caveats as fields — a correlation's floor, "this is correlation, not cause", a fit's out-of-sample score — so the client has them in hand.

`historian_status` answers *is this record complete*: samples received against samples persisted, what was dropped or spooled, when the last write landed. Every other answer rests on it.

Rules and models tools return documents (a rule's text, a model) or replays of what they would have found; nothing is armed or written into the project.

An argument a tool does not declare is refused, naming the unknown argument and listing what the tool takes.

Worked example: letting an assistant read the packaging line

1. Enable the **Industrial integration** module (see above).
2. **Recording ▸ Recorder Service…**: tick **Serve the data-access API** and **Serve the MCP surface, for AI assistants that query this plant's history**. Give the API a certificate (a routable address without TLS is refused). Press **Save config**, and start (or restart) the recorder service so it serves the new configuration.
3. **Administration ▸ Service Accounts…**: create an account named `Packaging assistant`, owner `process engineering`, role **Reporting — read only**, and tick **Mcp**. In **Readable signal prefixes** enter `Packaging/`. Leave the MCP limits blank. Press **Create account** and **Copy token** — the token is shown once and cannot be retrieved again.
4. In the AI client, add an MCP server of type HTTP at `https://plant-server:7443/api/v1/mcp` with the header `Authorization: Bearer <token>`.
5. Ask it, for example, "*Which signals on the checkweigher are there, and how did case weight behave during the last run?*" It will call `catalog_search`, then `semantic_executions` and `history_statistics` for `Packaging/Checkweigher/CaseWeight`.

To prove the route from a shell before involving a client (every request stands alone; there is no session):

```
curl -s https://plant-server:7443/api/v1/mcp \
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json" \
  -d '{"jsonrpc":"2.0","id":1,"method":"tools/call",
       "params":{"name":"catalog_search","arguments":{"search":"Checkweigher"}}}'
```

An account without the `Mcp` permission is answered *This account is not permitted to use MCP.*

Deploying it safely

- **Inside the plant**, point the client at the host directly.
- **Hosted outside**, put a reverse proxy in a DMZ that forwards **only** `/api/v1/mcp` and returns 404 for everything else, so REST, gRPC and the administration endpoints are unreachable from outside whatever the token permits. No Meelmo process should be the internet-facing listener.
- List the proxy under **Trusted reverse proxies** in **Recorder Service…** (or in `MEELMO_API_TRUSTED_PROXIES` for a standalone data-access server) and have it set `X-Forwarded-For`. Otherwise every client behind it shares the proxy's address, its failed-token limit and its name in the audit trail.
- A browser-based client must be named in the Origin allowlist, which is empty by default: ordinary MCP clients send no Origin and are admitted, and every web page is refused.
- Once an account can query through MCP, everything it is permitted to read can leave with the client's provider. Scope the account to what the assistant needs.

The server speaks the MCP revisions `2026-07-28`, `2025-06-18` and `2025-03-26`. A request without a version header is treated as `2025-03-26`; a request naming an unsupported revision is refused with the list of revisions this server serves.

The node's own MQTT broker

Instead of — or beside — publishing to a broker somebody else runs, the node can run one itself: a **governed unified namespace**. The line's panels, an MES gateway and the site's scripts connect to the node, subscribe to the plant's values, and publish into the subtrees they have been given. It is off by default.

What makes it governed

- **The plant's values are already there.** The node publishes each signal's current value on the topic its asset gives it — `acme/Dortmund/Line1/Furnace/Temperature` — retained, so a panel that connects mid-shift sees the state at once rather than waiting for a change. The topic levels are the asset model's path, so renaming a line renames its topics, and the levels are labelled with ISA-95's names (enterprise, site, area, line, cell) unless the site names its own.
- **Only the node publishes its own signals.** A client cannot publish onto one of them, whatever its rules say.
- **Nothing is granted by default.** Every client is one of the node's data accounts (or an anonymous client, only if the site admits them), and each **access rule** grants one subject `pub`, `sub` or `retain` under one topic filter.
- **Payload contracts.** A subtree can require a JSON object with named fields of given types, written on one line, or hold every message to a full JSON Schema (draft 2020-12) — ranges, patterns, enumerations, nested objects. A message that breaks a contract or a rule is refused and counted, never forwarded.
- **Everything is on the record.** The broker's rules are configuration, saved as a revision. Each refusal is written to the audit trail (at most once a minute per account and reason, with a count of the rest), and the broker's state — clients, subscriptions, topics, retained values and refusals — can be read at any time.

Switching it on

1. Open **Administration ▸ Historian** in the browser, panel **MQTT broker on this node**, or **Data ▸ Historian…** on the Desktop, tab **MQTT broker**.
2. Tick **Run the broker**. Leave **Listen address** empty to serve only this machine over plain MQTT on port 1883. To serve the plant network, give the address; the broker is then served over TLS on port 8883 with the node's site-PKI certificate — and not at all if the node has none. A broker is never served in the clear to another machine.
3. Write the access rules, one per line: `subject rights filter`. The filter is the rest of the line, so it may contain spaces.
4. Optionally, write payload contracts, one per line: `filter field:type …` (a filter with a space in it goes in double quotes), and schema contracts, as one JSON object from topic filter to JSON Schema.
5. Save. A rule or contract line that does not read is refused by its line number, and a schema that is not a valid draft 2020-12 schema is refused with what is wrong with it.

Broker fields

Field (browser / Desktop)	Default	Meaning
Run the broker / **Run the broker**	Off	Turns the broker on. In the browser the other fields appear only when it is ticked.
Listen address / **Listen address:**	This machine only	Empty serves this machine only, as plain MQTT. Any other address is served over TLS with the node's site-PKI certificate, and not at all without one; a warning says so while such an address is set.
MQTT port / **MQTT port:**	`1883`	Plain MQTT, served on this machine only. `0` for none.
MQTT over TLS port / **MQTT over TLS port:**	`8883`	Served whenever the node has a certificate from the site PKI. `0` for none.
Admit anonymous clients / **Admit anonymous clients**	Off	Off, every client is a node data account. On, an anonymous client gets only the rules written for `anonymous`, and a warning says nobody's name is on what they publish or read.
Publish this node's signals / **Publish this node's signals**	On	Each signal's current value, retained, on its asset's topic, so a client that subscribes later is handed the plant's state at once.
Enterprise level / **Enterprise level:**	Empty	The first topic level, before the asset model's own root. Empty starts at the asset model.
Namespace levels / **Namespace levels:**	`enterprise/site/area/line/cell`	What each level is called, top down, separated by `/`. Labels only: a topic's levels are its asset's path.
Root path / **Root path:**	Empty	An asset-path prefix removed from the topics, as for the OPC UA server. Must not begin or end with `/`.
Access rules / **Access rules:**	Empty	One rule per line: `account rights filter`. With no rule, every client is refused every topic, and the browser says so under the panel.
Payload contracts / **Payload contracts:**	Empty	One contract per line: `filter field:type …`, `?` after a type for an optional field, `strict` at the end to refuse other fields. Types: `number`, `integer`, `string`, `boolean`, `object`, `array`, `any`.

Field (browser / Desktop)	Default	Meaning
Schema contracts / **Schema contracts:**	Empty	One JSON object from topic filter to JSON Schema (draft 2020-12). See *Schema contracts* below.

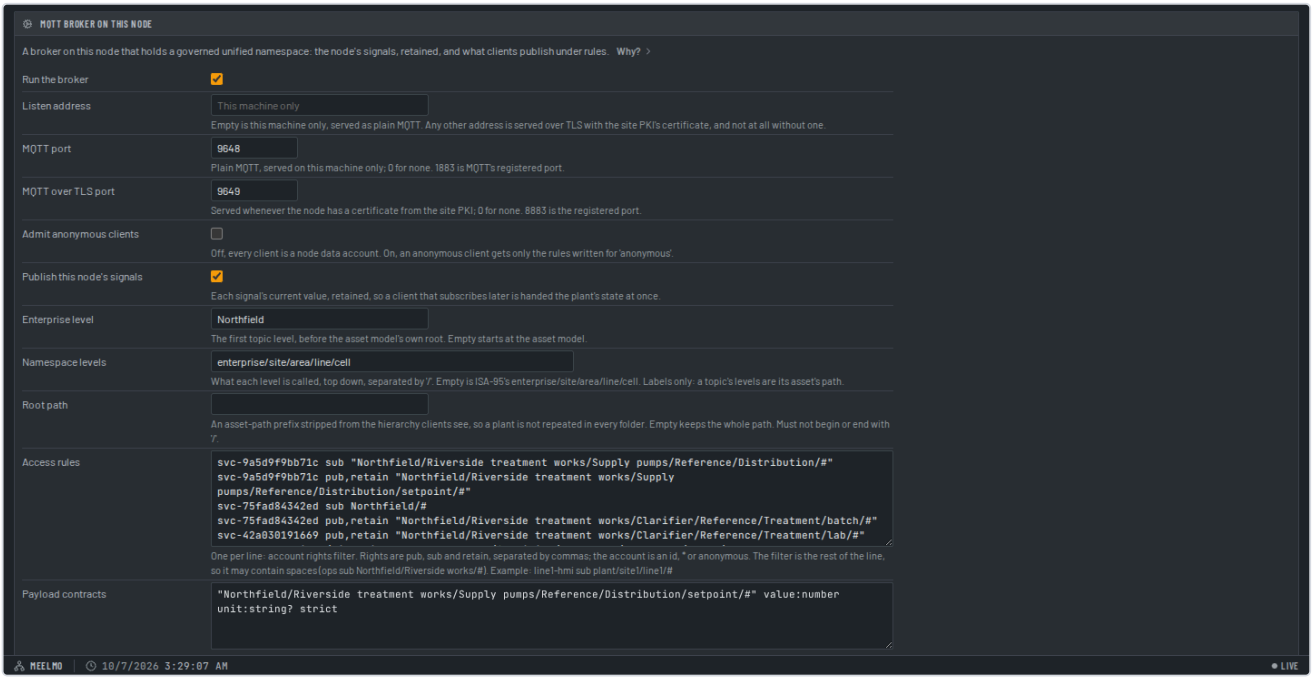
Topics with spaces in them

The namespace's levels are the asset model's names, and those have spaces in them — `Riverside treatment works`, `Supply pumps`. MQTT allows spaces in a topic level, and so do the rules and contracts; write the filter as the plant is called, never with `+` standing in for a word:

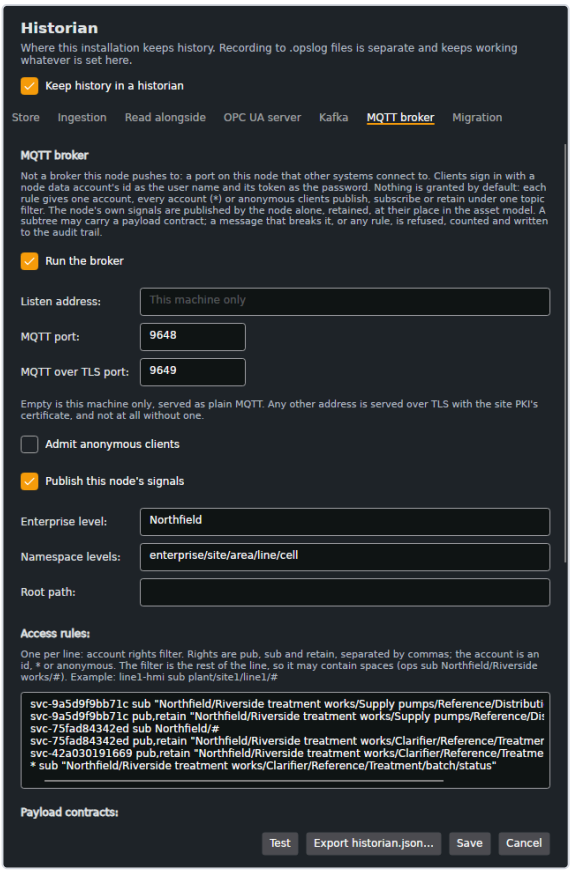
```
# A rule's filter is the rest of the line after the account and the rights.
pump-panel   sub,pub   Northfield/Riverside treatment works/Supply pumps/#

# A contract's fields follow its filter, so a filter with a space goes in double quotes.
"Northfield/Riverside treatment works/Supply pumps/#"   value:number unit:string?
```

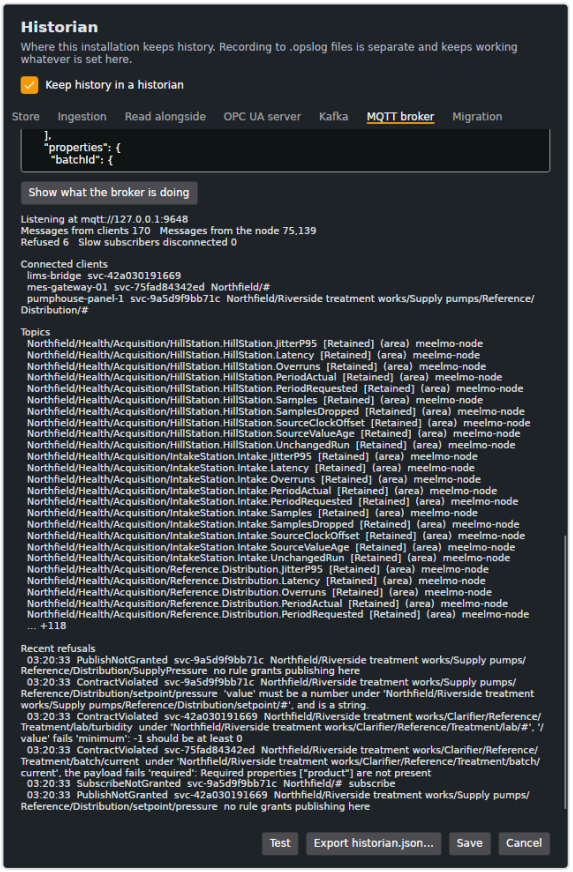
- A rule's filter may be written in double quotes too, and is written back that way when it has a space in it, so the two lists read alike. In a rule, quotes are needed only for a filter that begins or ends with a space, begins with a double quote or a comma, or begins with `pub`, `sub` or `retain` and a space; a double quote inside a quoted filter is written twice (`""`).
- A space is a character of the level like any other: `Riverside works` (two spaces) and `Riverside works` are different topics, and a rule for one grants nothing on the other.
- Schema contracts need nothing more: a JSON name is already quoted, so `"Northfield/Riverside treatment works/+/setpoint/#": { … }` is read as written.
- A contract line whose filter has a space but no quotes is refused with that advice, rather than read as a shorter filter with strange fields. The rights are written without spaces (`pub,sub`): `pub, sub` and `pub sub` are refused rather than read as the right `pub` on a filter beginning `sub`.



The MQTT broker panel of Administration ▸ Historian on the reference works: the broker run on this machine, the enterprise level Northfield under ISA-95's level names, six access rules for the pump house panel, the MES gateway and the LIMS bridge, a line contract on the setpoints and the schema contracts for the batch record and the laboratory results



The same settings on the Desktop: Data ▸ Historian..., MQTT broker tab



What the broker is doing, on a Desktop joined to the node and asked as the administrator signed in: its address, the messages each way, the connected clients and their subscriptions, the topics it holds and the refusals with their reasons

Schema contracts

When one line cannot say enough — a setpoint that must lie between 0 and 3000, a unit from a fixed list, a nested `limits` object — write the contract as a JSON Schema. **Schema contracts** is one JSON object; each name is a topic filter and each value the schema every payload under it must keep:

```
{
  "Dortmund/+/+/setpoint/#": {
    "type": "object",
    "required": ["value", "unit"],
    "properties": {
      "value": { "type": "number", "minimum": 0, "maximum": 3000 },
      "unit": { "$ref": "#/$defs/unit" }
    },
    "$defs": { "unit": { "enum": ["rpm", "m/min"] } }
  }
}
```

- **Draft 2020-12 only.** Leave out `$schema`, or write `https://json-schema.org/draft/2020-12/schema`.
- **Self-contained.** A `$ref` may point inside the schema (`#/$defs/…`) or to an `$id` the schema itself declares. A reference to another document — `https://…`, `other.json` — is refused when you save: the broker never fetches a schema from the network.
- **Checked once.** Each schema is validated against the draft 2020-12 meta-schema when you save, and compiled once when the broker starts; a message is checked against what was compiled. A schema is at most 64 KiB, and all of them together at most 512 KiB.
- **Bounded.** The payload is already capped by the broker's packet limit. A `pattern` or `patternProperties` regular expression gets 100 ms per value; one that runs longer refuses the message rather than holding the broker.
- **Both forms apply.** A line contract and a schema contract on the same subtree are both kept, as are contracts on overlapping filters. A schema may describe a payload that is not an object — an array, a number — which the line form cannot.
- **Refused like any other contract.** A message that breaks a schema is answered `PayloadFormatInvalid` with where it failed (`'/value' fails 'maximum'`), counted under `ContractViolated`, and kept in the recent refusals.

A schema that does not compile — possible only in a configuration edited by hand, since every screen checks it before saving — is logged when the broker starts, and nothing published under its filter is accepted until it is fixed: a contract that silently stopped applying would admit exactly the messages it was written to keep out.

The validator is JsonSchema.Net (MIT), held on its last MIT-licensed release; see `THIRD_PARTY_LICENSES.md`.

Worked example: the packaging line's panels and the MES

The packaging line's asset model is `Dortmund / Packaging / Line1 / …`, and the broker's **Enterprise level** is left empty, so topics start at `Dortmund`. The site wants the line's two HMI panels to read the line and write its setpoints, the MES gateway to read the whole site, and every account to see each line's status.

Data accounts (Administration ▸ Service accounts): `pack-hmi` and `mes-gateway`, each with its token.

Rules:

```
pack-hmi      sub           Dortmund/Packaging/Line1/#
pack-hmi      pub,retain    Dortmund/Packaging/Line1/setpoint/#
```

```
mes-gateway   sub           Dortmund/#
*             sub           Dortmund/+/+/status
```

Contract:

```
Dortmund/+/+/setpoint/#   value:number unit:string? strict
```

A panel connects with user name `pack-hmi` and its token as the password, and subscribes to `Dortmund/Packaging/Line1/#`. It is handed every signal's retained value at once. When it publishes `{"value": 42.5, "unit": "m/min"}` with retain to `Dortmund/Packaging/Line1/setpoint/speed`, the MES gateway receives it. When a script with the same account publishes `{"value": "fast"}` there, it is refused (`PayloadFormatInvalid` for an MQTT 5 client), nobody receives it, and the refusal appears under **Recent refusals** with the reason `ContractViolated`. When the panel tries `Dortmund/#`, the subscription is refused (`NotAuthorized`): its rule covers one line, not the site.

With `mosquitto_sub`, for testing from the node itself:

```
mosquitto_sub -h 127.0.0.1 -p 1883 -u pack-hmi -P '<token>' -t 'Dortmund/Packaging/Line1/#' -v
```

Limits

Every limit is a setting in the node's publication configuration and defaults to a value a line needs: 100 connections, 256 KiB per packet (refused from its header, before anything is allocated), 1,000 messages waiting for one subscriber (one that falls further behind is disconnected — the node never waits for a reader), a keep-alive of at most 300 seconds, 100,000 retained messages, 16 topic levels, 100 subscriptions per client and 30 failed sign-ins per address per minute. Shared subscriptions (`$share/…`) are refused as not supported.

Reading what it is doing

Show what the broker is doing on either settings screen, or `GET /api/v1/uns/broker` on the data API for an account with **Administer**, returns where the broker listens, its counters, the connected clients with their accounts and subscriptions, the topics with their level, retained flag and last publisher, and the refusals by reason.

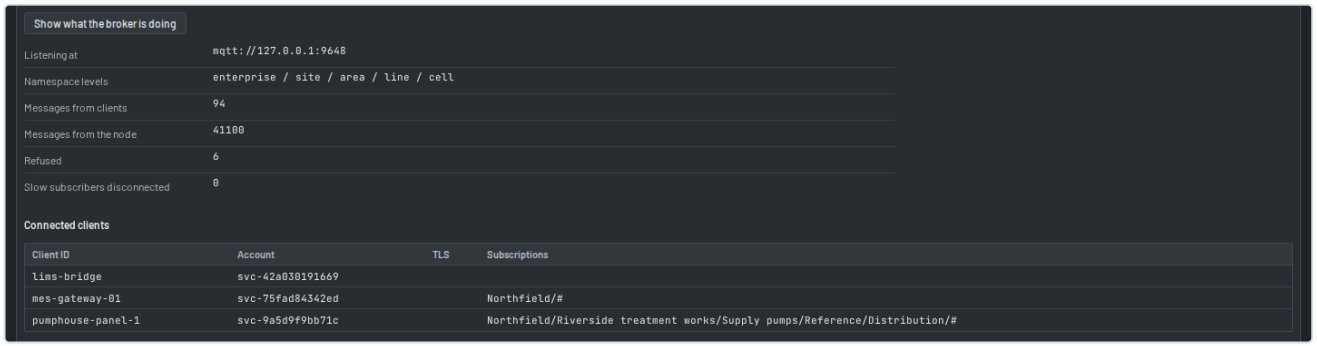
What the browser shows, under the button:

Item	Meaning
Listening at	The addresses the broker serves: `mqtt://<host>:1883` for plain MQTT, `mqtts://<host>:8883` over TLS.
Namespace levels	The level names in use, top down.
Messages from clients	Messages clients have published to the broker.
Messages from the node	Messages the node has published: its own signals.
Refused	Everything refused: messages, subscriptions and connections together. **Recent refusals** says which and why.
Slow subscribers disconnected	Clients that fell more than the limit of waiting messages behind and were disconnected, because the node never waits for a reader.

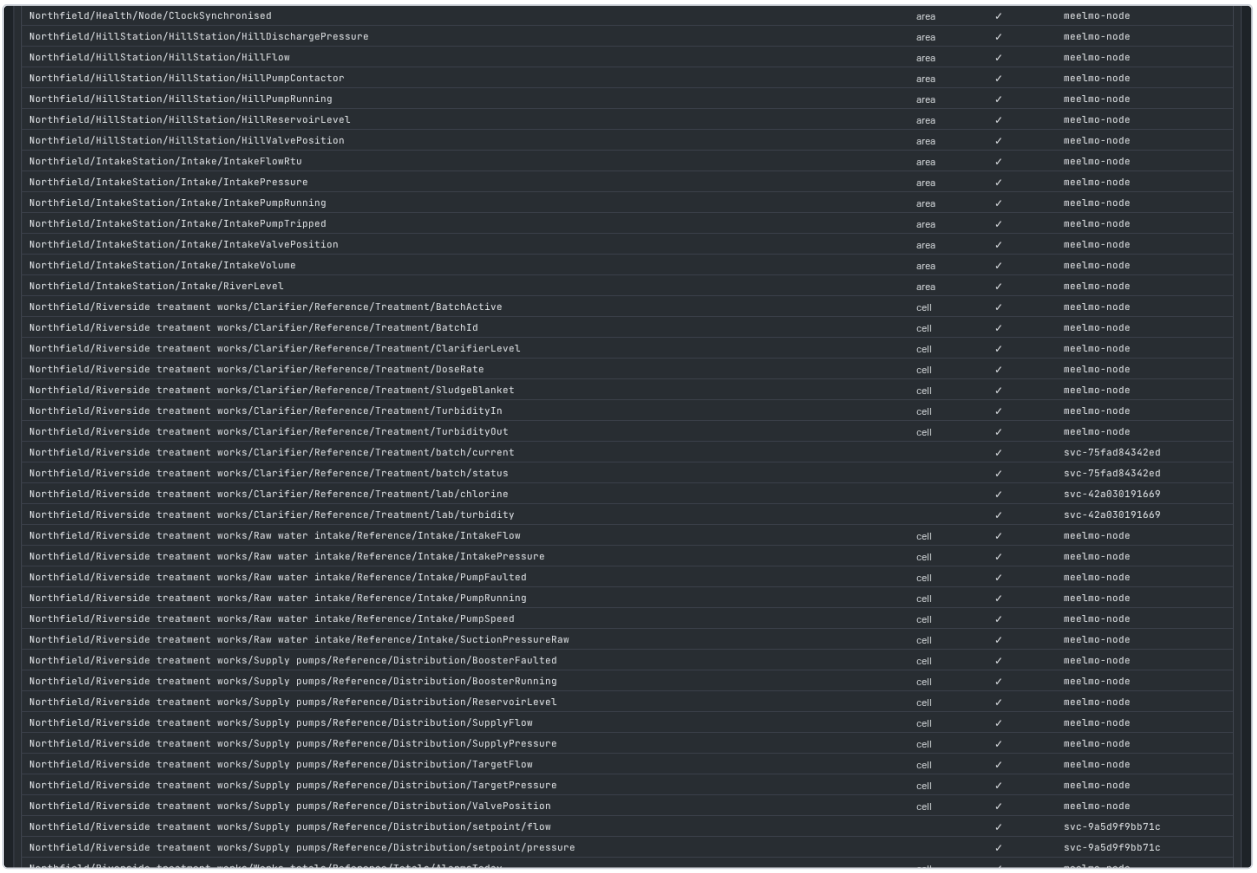
Table	Columns
Connected clients	**Client ID**, **Account** (the data account it signed in as), **TLS** (✓ when the connection is encrypted) and **Subscriptions** (its topic filters).
Topics	**Topic**, **Level** (the namespace level the topic sits at), **Retained** (✓ when the broker holds its last value) and **Last published by**.
Recent refusals	**Reason** (for example `ContractViolated`, `PublishNotGranted`, `SubscribeNotGranted`), **Account**, **Topic** and **Detail** (for a contract, where the payload failed).

An empty table says *None*. A broker that is switched off, or a node without a historian, shows the reason instead of the tables. The Desktop shows the same items as text in the **MQTT broker** tab.

On the Desktop the button asks the node as the person signed in, whether the Desktop holds the node or is joined to one elsewhere — not with the workstation's own data account, which may only read. Because the answer names the data accounts connected, the node shows it to a person whose role manages its users (an Administrator, as **Administer** is on the data API) and tells anyone else, in their own language, that they are not permitted to see it. Each look is on the node's audit trail under the person's name.

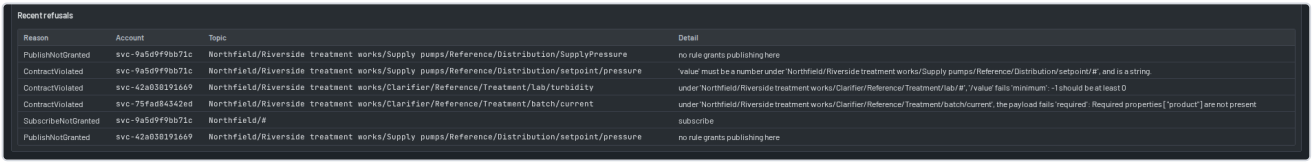


Show what the broker is doing: listening at mqtt://127.0.0.1, the namespace levels, messages from clients and from the node, refusals, and the three connected clients with their accounts and subscriptions



Northfield/Health/Node/ClockSynchronised	area	✓	meelmo-node
Northfield/HillStation/HillStation/HillDischargePressure	area	✓	meelmo-node
Northfield/HillStation/HillStation/HillFlow	area	✓	meelmo-node
Northfield/HillStation/HillStation/HillPumpContactor	area	✓	meelmo-node
Northfield/HillStation/HillStation/HillPumpRunning	area	✓	meelmo-node
Northfield/HillStation/HillStation/HillReservoirLevel	area	✓	meelmo-node
Northfield/HillStation/HillStation/HillValvePosition	area	✓	meelmo-node
Northfield/IntakeStation/Intake/IntakeFlowRtu	area	✓	meelmo-node
Northfield/IntakeStation/Intake/IntakePressure	area	✓	meelmo-node
Northfield/IntakeStation/Intake/IntakePumpRunning	area	✓	meelmo-node
Northfield/IntakeStation/Intake/IntakePumpTripped	area	✓	meelmo-node
Northfield/IntakeStation/Intake/IntakeValvePosition	area	✓	meelmo-node
Northfield/IntakeStation/Intake/IntakeVolume	area	✓	meelmo-node
Northfield/IntakeStation/Intake/RiverLevel	area	✓	meelmo-node
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/BatchActive	cell	✓	meelmo-node
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/BatchId	cell	✓	meelmo-node
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/ClarifierLevel	cell	✓	meelmo-node
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/DoseRate	cell	✓	meelmo-node
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/SludgeBlanket	cell	✓	meelmo-node
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/TurbidityIn	cell	✓	meelmo-node
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/TurbidityOut	cell	✓	meelmo-node
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/batch/current		✓	svc-75fad84342ed
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/batch/status		✓	svc-75fad84342ed
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/lab/chlorine		✓	svc-42a030191669
Northfield/Riverside treatment works/Clarifier/Reference/Treatment/lab/turbidity		✓	svc-42a030191669
Northfield/Riverside treatment works/Raw water intake/Reference/Intake/IntakeFlow	cell	✓	meelmo-node
Northfield/Riverside treatment works/Raw water intake/Reference/Intake/IntakePressure	cell	✓	meelmo-node
Northfield/Riverside treatment works/Raw water intake/Reference/Intake/PumpFaulted	cell	✓	meelmo-node
Northfield/Riverside treatment works/Raw water intake/Reference/Intake/PumpRunning	cell	✓	meelmo-node
Northfield/Riverside treatment works/Raw water intake/Reference/Intake/PumpSpeed	cell	✓	meelmo-node
Northfield/Riverside treatment works/Raw water intake/Reference/Intake/SuctionPressureRaw	cell	✓	meelmo-node
Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/BoosterFaulted	cell	✓	meelmo-node
Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/BoosterRunning	cell	✓	meelmo-node
Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/ReservoirLevel	cell	✓	meelmo-node
Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/SupplyFlow	cell	✓	meelmo-node
Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/SupplyPressure	cell	✓	meelmo-node
Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/TargetFlow	cell	✓	meelmo-node
Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/TargetPressure	cell	✓	meelmo-node
Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/ValvePosition	cell	✓	meelmo-node
Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/setpoint/flow		✓	svc-9a5d9f9bb71c
Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/setpoint/pressure		✓	svc-9a5d9f9bb71c

Part of the topics: the node's own health and plant signals, retained and published by the node, beside the setpoints, batch record and laboratory results the clients keep under the subtrees their rules give them



Recent refusals

Reason	Account	Topic	Detail
PublishNotGranted	svc-9a5d9f9bb71c	Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/SupplyPressure	no rule grants publishing here
ContractViolated	svc-9a5d9f9bb71c	Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/setpoint/pressure	'value' must be a number under 'Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/setpoint/#', and is a string.
ContractViolated	svc-42a030191669	Northfield/Riverside treatment works/Clarifier/Reference/Treatment/lab/turbidity	under 'Northfield/Riverside treatment works/Clarifier/Reference/Treatment/lab/#', '/value' fails 'minimum' : -1 should be at least 0
ContractViolated	svc-75fad84342ed	Northfield/Riverside treatment works/Clarifier/Reference/Treatment/batch/current	under 'Northfield/Riverside treatment works/Clarifier/Reference/Treatment/batch/current', the payload fails 'required' : Required properties ["product"] are not present
SubscribeNotGranted	svc-9a5d9f9bb71c	Northfield/#	subscribe
PublishNotGranted	svc-42a030191669	Northfield/Riverside treatment works/Supply pumps/Reference/Distribution/setpoint/pressure	no rule grants publishing here

Recent refusals: a word in a setpoint, a batch record without its product and a turbidity below zero, each refused by its contract (ContractViolated, with where it failed); a subscription to the whole site (SubscribeNotGranted); and two publishes no rule grants — the panel onto the node's own SupplyPressure and the laboratory bridge onto a setpoint (PublishNotGranted)

Connectors are not part of this module

The automation connectors — outbound HTTP requests, an inbound webhook, and local files — belong to automation, which comes with the Web HMI module. They let a rule tell another system something or be started by one, and they are unaffected by whether this module is licensed. Publication in this chapter is continuous and sample-by-sample; a connector is called by a rule, once, when the rule decides to.

Troubleshooting

Symptom	Cause and remedy
Nothing is published and the Desktop says the historian is not on.	Publication is a historian sink. Tick **Keep history in a historian** (any store).

Symptom	Cause and remedy
Publication transport '…' is not one this application can carry.	The configuration names an unknown transport, or the module is off on this node. Choose **MQTT** or **Sparkplug B** and check **Administration ▸ Modules**.
The publisher connects and is disconnected in a loop.	Another client uses the same **Client id**. Give each publisher its own.
The broker refuses the connection.	Check host, port, TLS and credentials. A broker that disabled MQTT 3.1.1 is retried on 5.0 automatically.
Subscribers see stale values after the line stopped.	The retain flag is on. Turn off **Ask the broker to keep the last value per topic** and clear retained messages on the broker.
Samples are counted as refused.	The queue is full because the broker is slow or unreachable. Acquisition is not affected. Fix the broker link; raise **Queue capacity** only for short, expected gaps.
A Sparkplug group id may not contain '+', '/' or '#'.	Group and edge node ids are topic levels. Use a single level such as `PlantA`.
A Sparkplug host shows metrics it cannot resolve.	It missed the birth. It can request a rebirth with `Node Control/Rebirth`; the node answers at most every 30 seconds.
New signals do not appear on the Sparkplug host.	They are dropped until the next birth; one follows within 30 seconds.
The MCP route returns 404.	One of the three gates is not met: the switch, the licence or the module. The host's log names which.
This account is not permitted to use MCP.	Tick **Mcp** for that service account.
An assistant reports its answer was shortened.	**Samples per tool call** was reached. Ask a narrower question, or raise the limit for that account.
A client cannot connect to the node's broker.	Its user name must be a data account's id and its password that account's token. On another machine, it must use TLS on port 8883 — the plain port serves this machine only. A keep-alive of 0, or over 300 seconds, is refused.
A client is connected and receives nothing.	No rule's filter covers its subscription whole. **Recent refusals** shows `SubscribeNotGranted`; widen the rule or narrow the subscription.
A publish to the broker reaches nobody.	It was refused: see **Recent refusals** for `PublishNotGranted`, `RetainNotGranted`, `ContractViolated` or `NodeOwnedTopic`. An MQTT 3.1.1 client is not told; an MQTT 5 client gets the reason code.
The broker serves nobody and says the address is not loopback.	The node has no site-PKI certificate, and a broker off this machine is never served in the clear. Adopt the site PKI, or bind 127.0.0.1.
All assistants behind a proxy are refused together.	The proxy is not trusted, so they share one address's failed-token limit. Add it to **Trusted reverse proxies**.

Models

> **Module: Models (`models`).** This chapter describes fitting models to recorded history and estimating from them what the plant does not measure continuously. It is present only on installations licensed for the Models module. Without it, the Desktop's Models Studio and the browser's Models page say so instead of offering a form. Correlation and investigation without models are part of the base product; see the *Analysis* chapter.

What models are for

Every plant knows something it cannot measure continuously: a laboratory assay taken three times a shift, a flow nobody instrumented, a quality figure that arrives hours after the product did. A **soft sensor** estimates that value in between — the measurements that do exist are the answers, and the signals recorded alongside them are the question.

The Models module does this in four steps, and the order matters:

1. **What moves with it?** Every signal the project publishes is correlated against the target at a handful of delays. Hundreds of signals become a shortlist of at most twelve.
2. **Choose.** You pick inputs from the shortlist, by hand. A ranking cannot tell a cause from a consequence; you can.
3. **Fit.** Ridge regression gives an equation you can read, and a score measured on a period the fit never saw.
4. **Estimate**, only once the score says the model is usable.

Two further uses build on a fitted model:

- **Watching** what the model gets wrong: an alarm on a relationship breaking, rather than on one number getting large.
- **Forecasting**: the same fit, answering about the value some time ahead.

Everything here is read-only against the plant. A fit returns a model document; nothing is armed or written into the project until a person puts it there through a configuration revision.

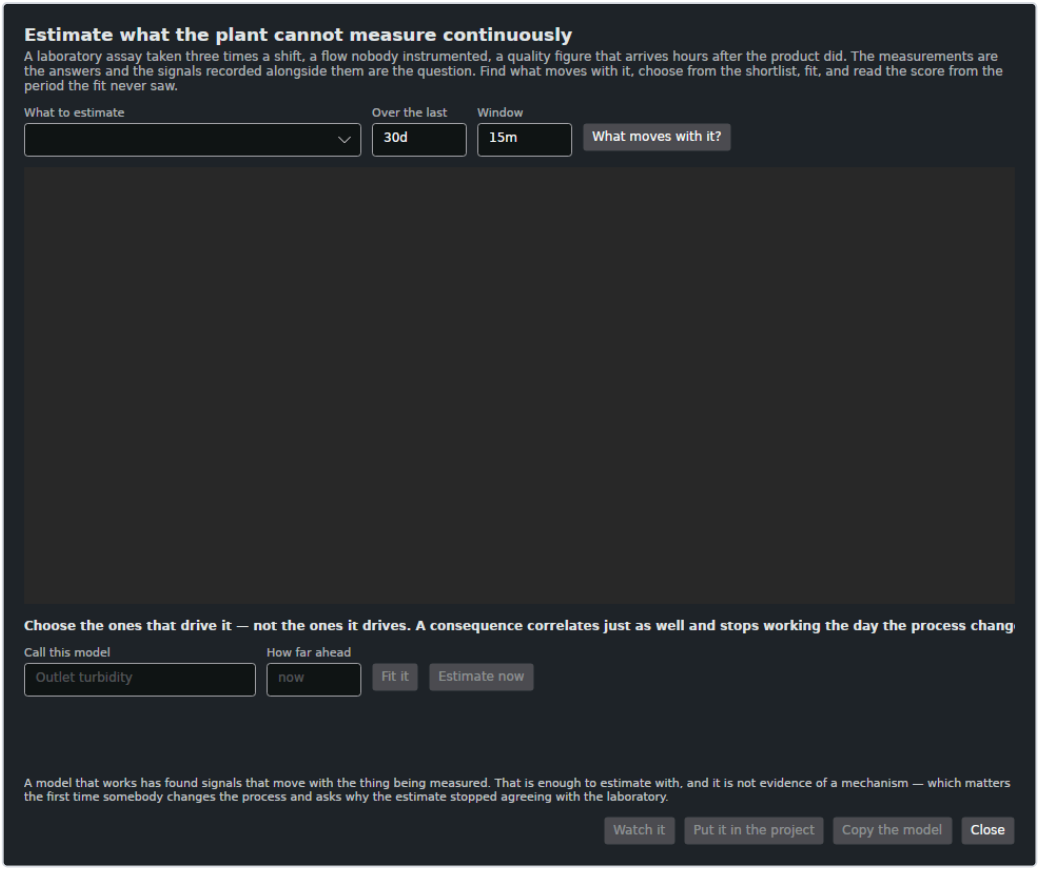
Where to find it

Surface	Path	Who
Desktop	**Data ▸ Models Studio…**	Anyone who may read history, on a Desktop attached to a node with the module
Browser (with the Web HMI module)	**Analytics ▸ Models**: the **Models** tab beside **What changed**, **What relates** and **Compare runs**	Anyone who may read history
Data-access API	`POST /models/candidates`, `/models/fit`, `/models/estimate`, `/models/watch/tune`, `/models/watch/replay`	A service account that may read history
Desktop, models at the edge	**Data ▸ Models at the Edge…**	Anyone who may read the node's catalogue; changing needs the permission to configure signals

Surface	Path	Who
Browser, models at the edge	**Edge models** beside **Models**	The same; **Configure a model signal** needs the permission to configure signals and to change configuration
Data-access API, models at the edge	`GET /models/registry`, `GET /models/inference`, `GET /models/signals`, `POST /models/registry/{model}`, `POST /models/deployments/{signal}`, `POST /models/deployments/{signal}/rollback`, `POST /project/model-signals`	Reading needs the catalogue permission; changing needs to configure signals, or Administer (a model signal into the project: to configure signals, as a person the node lets draft configuration)

The same arithmetic runs behind all three. A model that reads a signal your account may not see is refused rather than fitted without it, and an estimate whose input has no samples behind it is refused rather than guessed.

The Models Studio on the Desktop



The Desktop Models Studio estimating Packaging/Checkweigher/CaseWeight: the shortlist of signals that move with it, three chosen as inputs, and the fitted model with its equation and its score on the period it never saw

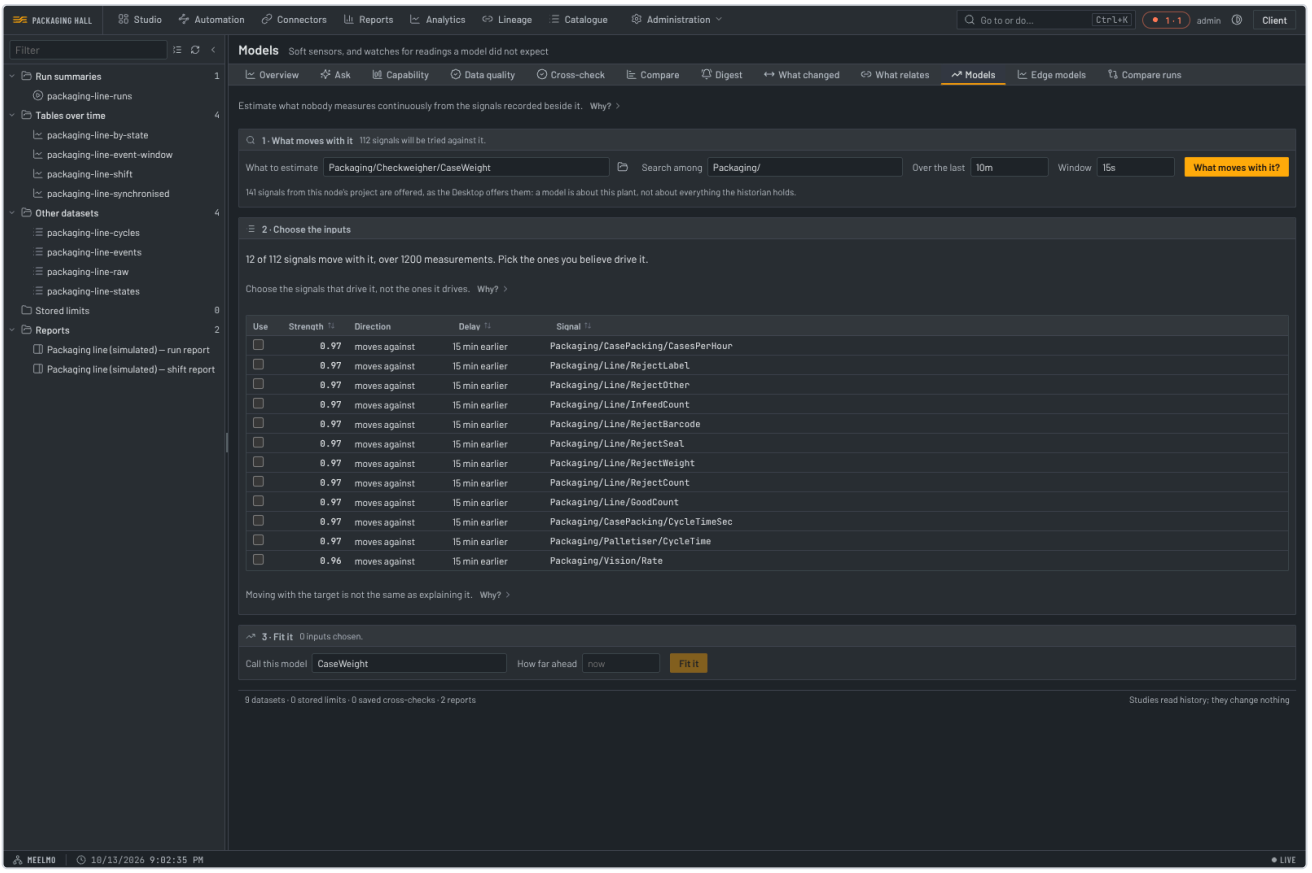
The window reads top to bottom in the order of the four steps.

Element	Purpose
What to estimate	The target signal, chosen from the project's signals.
Over the last	The period of history to learn from, such as `30d` (the default) or `12h`.

Element	Purpose
Window	The window each input is averaged over, default `15m`. One window for every input: it describes how quickly the target responds.
What moves with it?	Runs step 1 and fills the shortlist.
Shortlist	One row per candidate: strength (0 to 1), direction (**moves with** or **moves against**), the delay at which it looked best (**no delay**, or the delay followed by *earlier*), and the signal path.
Choose the ones that drive it — not the ones it drives.	The reminder above the fit controls. Select one or more rows (Ctrl-click or Shift-click).
Call this model	The model's name, for example `Case weight estimate`.
How far ahead	Empty (showing `now`) for a soft sensor. A duration such as `30m` or `2h` makes it a forecast.
Fit it	Step 3. Enabled once at least one row is selected.
Estimate now	Step 4. Enabled only when the fit is usable.
Equation and verdict	The fitted equation and a sentence saying what it is worth.
Watch it	Opens *Where to set the watch*. Enabled only when the fit is usable.
Put it in the project	Shown when the Desktop can change the node's project. Adds the model to the project as a configuration change, even if the score is poor — the score travels with it and shows in the diff.
Copy the model	Puts the model document on the clipboard.
Close	Closes the window. Nothing is kept unless you copied it or put it in the project.

The Models page in the browser

The browser page offers the same four steps as numbered panels, and the watch after them.



The browser Models page after What moves with it?: the target Packaging/Checkweigher/CaseWeight, the period and window, and the shortlist of candidates ranked by strength with their direction and delay, each with a Use tick box

1 · What moves with it

Field	Purpose
What to estimate	Type the signal path, or press the folder button (**Choose from the node's signals**) to pick it.
Search among	Narrow the candidates to part of a path, for example `Packaging/Loader`. Empty tries every signal.
Over the last	Default `30d`.
Window	Default `15m`.
What moves with it?	Runs the search.

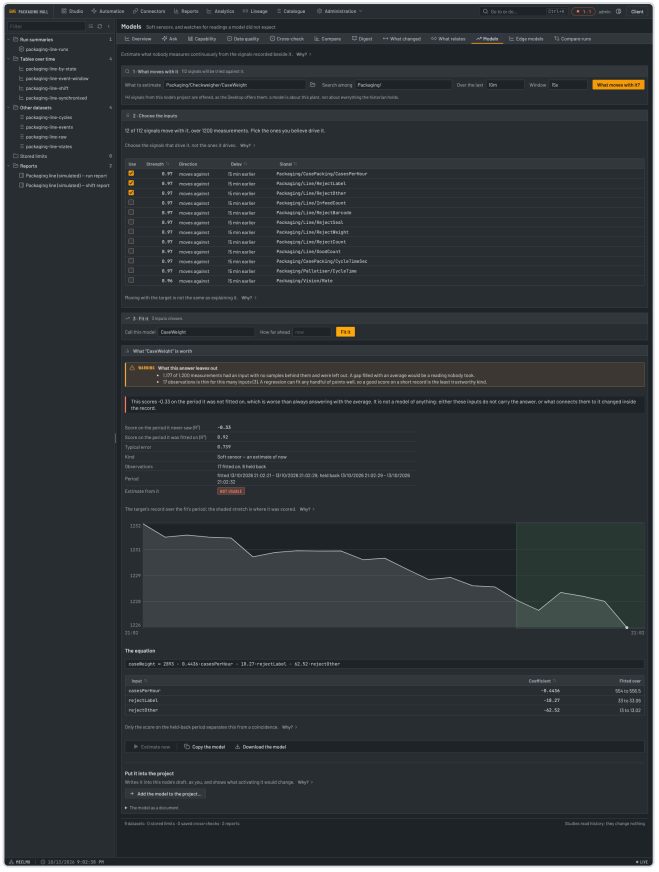
The line under the panel says how many signals will be tried, for example *141 signals from this node's project are offered, as the Desktop offers them: a model is about this plant, not about everything the historian holds.* A licence limits how many signals one question may try; if the search is too wide, the page says so and asks you to narrow **Search among**.

2 · Choose the inputs

The shortlist, strongest first: **Signal**, **Strength**, **Direction**, **Delay**, and a **Use** tick box per row. The summary line reads, for example, *5 of 141 signals move with it, over 1,200 measurements. Pick the ones you believe drive it.* Where the Studio Assistant is licensed and configured, an extra column **Advice — not a measurement** may carry a short note per row, such as *downstream*, *same measurement* or *coincidence*. The notes never reorder the list and never change a number.

3 · Fit it

Call this model, **How far ahead**, and **Fit it**. The result appears as *What "<name>" is worth*:



The fitted model in the browser: the verdict, the facts table with the score on the period it never saw and on the period it was fitted on, typical error, kind, observations, period and whether it is usable; the target's record with the held-back stretch shaded; the equation and each input's coefficient and fitted range; and the Estimate now, Copy the model and Download the model buttons

Fact	Meaning
Score on the period it never saw (R^2)	The number to read. Measured on the held-back stretch.
Score on the period it was fitted on (R^2)	For comparison. A large gap between the two means memorising.
Typical error	How wrong an estimate typically is, in the target's units.
Kind	*Soft sensor — an estimate of now*, or *Forecast, 30 min ahead*.
Observations	For example *900 fitted on, 300 held back*. The last quarter of the record is held back.
Period	The fitted and held-back time ranges.
Doing nothing would be wrong by	Forecasts only: the error of carrying the last known value forward.
Estimate from it	**Usable** or **Not usable**.

Below the facts: a chart of the target over the fit's period with the held-back stretch shaded (*Held back — scored here*); **The equation**; a table of each **Input**, its **Coefficient** and the range it was **Fitted over**; and the buttons **Estimate now**, **Copy the model** and **Download the model**. **The model as a document** folds open the full model text.

Under the result, **Put it into the project** keeps the model in the project from the browser, in two presses with a review between them:

Control	Does
Add the model to the project…	Writes the model into the node's configuration draft, as you, and shows **What activating the draft would change**. Nothing is activated yet
The headline	*Adds the model "…" to the project.*, or *Replaces the model "…" the plant runs now — a refit.*, with the revision it was measured against
The settings that change	Each with **Change**, **Setting**, **Now** and **After**
Warnings	Anything else the draft already holds from somebody else ("Activating it activates that too"), and anything the node's validation refuses (*The node would not activate this, and says why:*). A refusal greys **Activate**
Why, for the revision history	The sentence the revision keeps
Activate	Makes it a new revision: *Activated. The plant runs it from revision N.*
Take it back out	Removes only this entry from the draft, because the draft may hold somebody else's edits

It is the same review used for a rule from **What changed** and for a glossary term. It is offered whatever the fit scored: a site may keep a model that scored badly, and the score travels in the document. Somebody the node does not let change its configuration reads *You are not permitted to change this node's configuration, so this cannot be put into the project from here. Copy or download it for somebody who is.* and keeps **Copy the model** and **Download the model**. On the Desktop, **Put it in the project** does it in one step.

Estimating what is not measured: step by step

The lag is not a detail

A plant has transport in it. Coagulant dosed now reaches a turbidity meter twenty minutes downstream; correlating this minute's dose against this minute's turbidity compares two different batches of water, and the coefficient comes out near zero — "dose does not matter", the opposite of the truth.

So every candidate is tried at a fixed set of delays — none, 5 minutes, 15 minutes, 30 minutes, 1, 2 and 4 hours — and the shortlist reports the delay each looked best at. Each chosen input keeps that delay in the model. The delays are a short list rather than a fine search on purpose: a fine sweep finds the delay that fits this record best, which on a noisy record is often not the one that is real. If your engineers know the residence time, check the shortlist's delay against it.

Choosing inputs

> Choose the ones that **drive** the target, not the ones it drives. A consequence correlates just as well as a cause — often better — and a model built from consequences works well until the day somebody changes the process.

Three entries that are near-copies of each other (one instrument under several names, or a raw value and its filtered copy) are normal; pick one. Then check the signs in the equation against what you know: a wrong sign is the commonest way a regression admits it found a correlation rather than a mechanism.

Reading the score

The score is **R² on the period held back** — the last quarter of the record, never fitted on. Holding back random rows instead would be close to meaningless on process data: consecutive readings a minute apart are nearly the same reading, so a randomly held-back row is all but present in its neighbours. Models validated that way report 0.9 and deliver 0.3.

What comes back	Verdict sentence	What to do
Negative, or near zero	*This scores −0.12 on the period it was not fitted on, which is worse than always answering with the average.*	Not a model of anything. Different inputs, or check whether the process changed inside the record.
High in sample, low out of sample	*It explains 96% of the period it was fitted on and 68% of the period it was not. That gap is memorising rather than learning.*	Fewer inputs, or a longer record.
Good out of sample	*It accounts for 91% of the variation on the period it never saw, and is wrong by about 0.18 on average.*	Usable. The second figure is what to weigh an estimate by.

Negative scores are reported negative, not clamped at zero.

Estimating

Estimate now returns a value and an interval: *Estimated 12.41, between 12.18 and 12.64.* If an input is outside the range the model was fitted over, the answer says so and names it — *Packaging/Loader/Rate is outside the range this model was fitted over, so this is an extrapolation.* A straight line answers confidently anywhere; it was only ever measured where the data was.

> **A model that works has found signals that move with the thing being measured.** That is enough to estimate with, and it is not evidence of a mechanism.

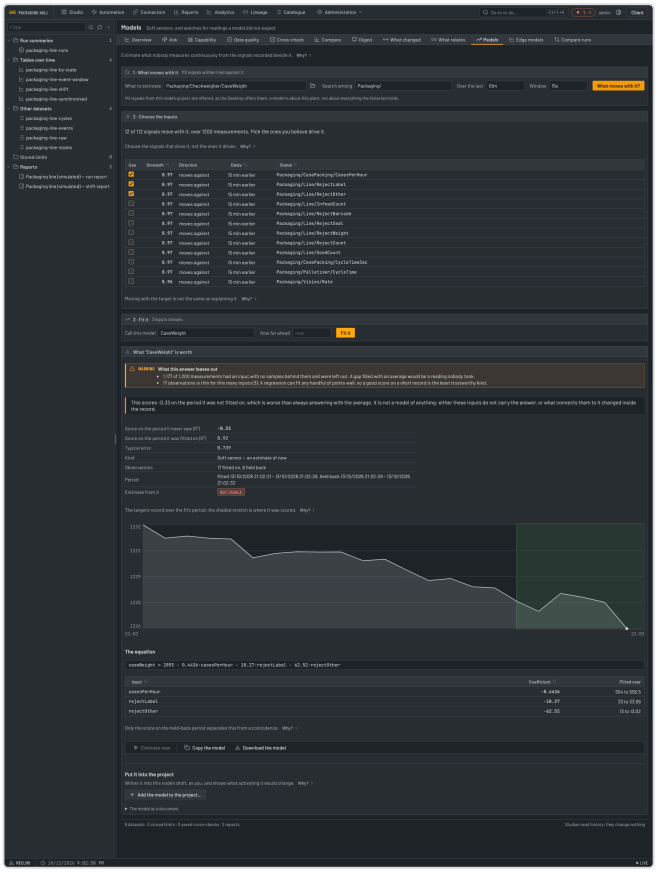
Worked example: estimating case weight on the packaging line

The checkweigher on the demonstration packaging line measures `Packaging/Checkweigher/CaseWeight`, but suppose a second line has no checkweigher and the question is whether case weight can be estimated from the loader's signals. Fit on the line that has one.

1. Open **Models** in the browser (or **Data ▸ Models Studio…** on the Desktop).
2. **What to estimate:** `Packaging/Checkweigher/CaseWeight`. **Search among:** `Packaging/Loader`. **Over the last:** `30d`. **Window:** `15m`.
3. Press **What moves with it?**. The shortlist ranks the loader's signals — for example `Packaging/Loader/Rate`, `Packaging/Loader/CollatorBacklog`, `Packaging/Loader/PouchInfeedRate` and `Packaging/Loader/PouchesPerCase` — with a strength, a direction and a delay for each.
4. Choose inputs that plausibly **drive** case weight. `PouchesPerCase` and the infeed rate act upstream of the checkweigher; a signal that is computed from the checkweigher's own reading would be a consequence and should be left out.
5. **Call this model:** `Case weight estimate`. Leave **How far ahead** empty.
6. Press **Fit it** and read, in this order: **Estimate from it**, the **Score on the period it never saw**, the gap to the **Score on the period it was fitted on**, and the **Typical error** against the checkweigher's tolerance.
7. Check each coefficient's sign and its **Fitted over** range.

8. If it is usable, press **Estimate now**, then **Copy the model** (or, on the Desktop, **Put it in the project**).

What one real capture of the demonstration line looked like, fitted over five minutes of simulated data with three inputs:



An earlier capture of the fitted case-weight model on the demonstration line. It scores 0.96 on the period it was fitted on and 0.68 on the period it never saw, with a typical error of 4.88 over 461 fitted and 154 held-back observations; the page warns that the gap is memorising rather than learning, and that 585 measurements whose inputs had no samples were left out

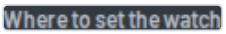
Read it as the page does. It is marked **Usable**, but the verdict says the model explains 96% of the fitted period and 68% of the held-back one: a gap that size is memorising. The remedy the page names — fewer inputs, or a longer record — is the right one: fit over `30d` rather than minutes, or drop the weakest input. The first note also matters: 585 of the 1,200 measurements had an input with no samples behind them and were left out — a gap is never filled with an average.

Watching what the model gets wrong

A heat exchanger fouls. Nothing goes out of range — the outlet temperature stays inside every limit — so no alarm on any single signal fires, for weeks. What changes immediately is that the outlet is no longer what the inlet and the flow say it should be. That gap is a **residual**, and a watch on it fires on a relationship breaking.

Where to set the watch

On the Desktop press **Watch it**; in the browser, the panel **Where to set the watch** follows the fit.



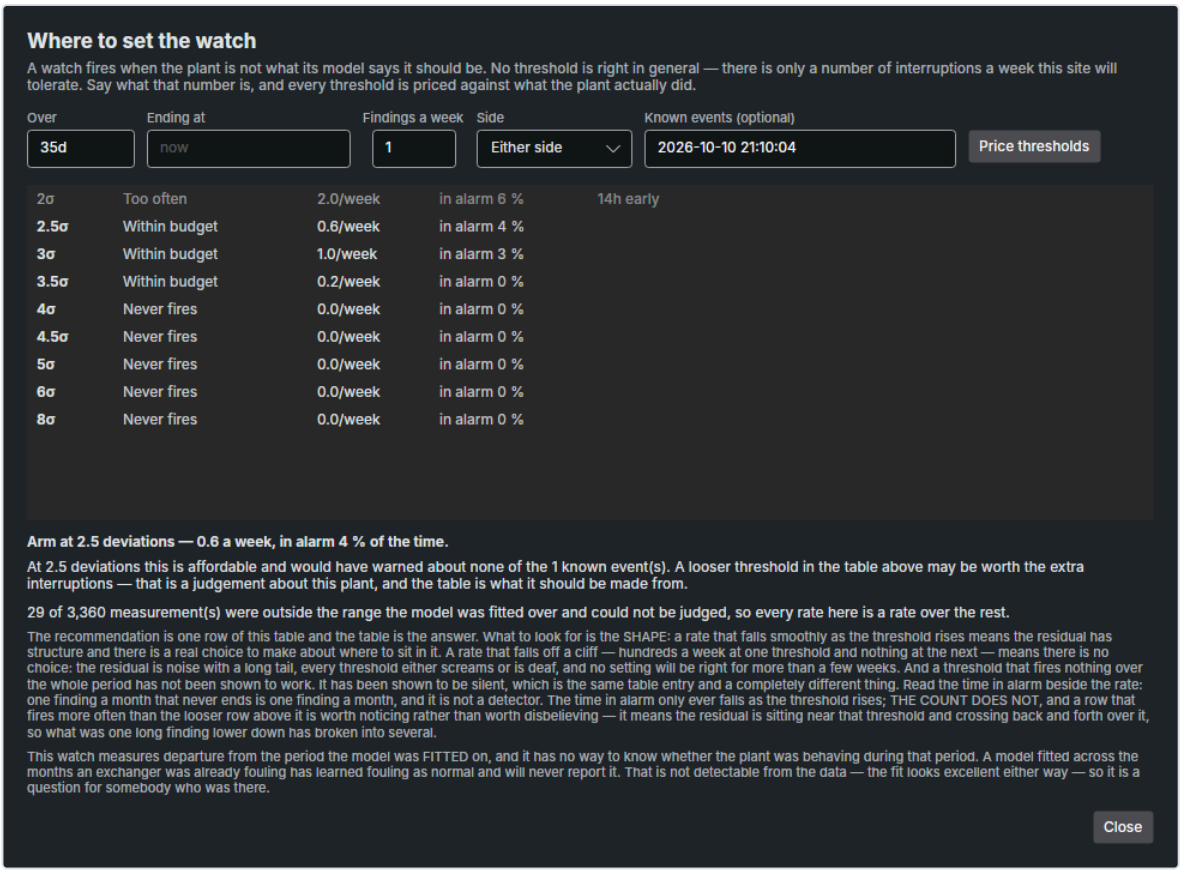
Every threshold priced against the record in the browser: the period, findings a week, side and known events, then a table of thresholds with their verdict, rate a week, time in alarm and known events caught

Field	Default	Meaning
Over	`90d`	The period of history to price thresholds against.
Ending at	now	The end of that period.
Findings a week	`1`	How many interruptions a week this site will tolerate.
Side	**Either side**	Also **Above the model** or **Below the model**.
Known events (optional)	—	Dates and times when something actually went wrong, comma separated.
Price thresholds	—	Runs the pricing.

The answer is a table, loosest threshold first:

Column	Meaning
Threshold	In deviations of the residual, for example `2.5σ`.
Verdict	**Too often**, **Within budget**, **Never fires**, or **Standing alarm**; the recommended row is marked *recommended*.
Rate	Findings a week, for example `0.6 a week`.
Time in alarm	The share of the period the watch would have been in alarm.
Known events caught	For example `1 of 1 · 14 h early`.
Replay (browser)	Replays that threshold over the period and lists each finding: **Began**, **Lasted**, **Peak**, **Measured against expected**.

Above the table is the recommendation, for example *Arm at 2.5 deviations — 0.6 a week, in alarm 4% of the time.*



The Desktop's Where to set the watch window for a model on the demonstration line, priced over 35 days at one finding a week with one known event: 2σ fires too often, 2.5σ to 3.5σ are within budget, and 4σ and above never fire

Reading the table

- **Read the shape, not the recommendation.** A rate that falls smoothly as the threshold rises means there is a real choice about where to sit. A rate that falls off a cliff — hundreds a week at one threshold and nothing at the next — means the residual is noise with a long tail and no setting will be right for long.
- **A threshold that never fires has not been shown to work.** It has been shown to be silent.
- **Time in alarm travels beside the rate.** A residual that never returns opens one finding and never closes it; one finding a month that never ends is not a detector. A threshold in alarm more than a fifth of the period is not counted as affordable.
- **Known events price the other half.** Any false-alarm budget can be met by raising the threshold; known events show what that costs. A finding that begins after the event does not count as a warning.

The watch refuses three things: a model that does not estimate (every residual would be the model being wrong); measurements taken while an input is outside the fitted range (excluded and counted, with the input named — the watch was blind then); and a model reading a signal the account may not see.

> **A finding is not a fault.** A recalibrated transmitter, a changed setpoint, a new feedstock or a season the fit never saw produce exactly what a developing fault produces. A residual that sits on one side and grows is the process drifting; one that swings both ways is usually the measurement. And a model fitted across months when the exchanger was already fouling has learned fouling as normal and will never report it.

A replay judges every measurement in the period and is bounded at 100,000 measurements of the target; a longer period is refused with the length that would fit. Nothing here arms a watch: a watch is configuration and goes into the project through a revision.

Worked example: a watch on case weight

With a usable `Case weight estimate`:

1. Open the watch (**Watch it** on the Desktop).
2. **Over:** `35d`. **Findings a week:** `1`. **Side: Either side**.
3. **Known events (optional):** the date and time a pouch-count fault was found on the line, for example `2026-09-19 20:32:51`.
4. **Price thresholds**.
5. Read the table from the top. In the capture above, `2σ` fires too often (2.0 a week), `2.5σ` is within budget (0.6 a week, in alarm 4%), and from `4σ` it never fires. The recommendation is 2.5 deviations — and the page adds that at that threshold it would have warned about none of the one known event, so a looser row may be worth its extra interruptions. That is a judgement about the plant, made from the table.
6. The page also reports how many measurements were outside the fitted range and could not be judged (in the capture, 29 of 3,360).

Forecasting: How far ahead

Put a duration in **How far ahead** — `30m`, `2h` — and the model answers about then, from what is known now. It is the same arithmetic: the horizon is added to every input's delay, so the fit only ever sees what would have been available when the prediction was due. You cannot accidentally train a forecast on the answer.

> **For a forecast, R² is not the number to read.** "It will be what it is now" is a forecast, and on a slowly wandering signal a very good one. R² is measured against the period's average, which is a much weaker opponent than the present value.

So every forecast is also scored against **carrying the last known value forward**, over the same held-back period:

- *Carrying the last known value forward would be wrong by about 0.35, so this is genuinely adding something.* — the fit **beats doing nothing**.
- *It is wrong by about 0.9, against 0.12 for simply carrying the last known value forward — so it is no better than doing nothing.* — **Estimate now** stays disabled, whatever the R².

A soft sensor has no such comparison, because the target was not measured at that instant; the field is absent rather than zero.

For the packaging line: target `Packaging/Conveyors/BufferLoaderToCheckweigher`, inputs from the loader and checkweigher, **How far ahead** `30m`. A forecast of buffer level half an hour out is useful only if it beats "the buffer will be where it is now"; the **Doing nothing would be wrong by** fact says whether it does.

Keeping a model

Copy the model puts the model document on the clipboard; **Download the model** (browser) saves it as a file; **Add the model to the project…** (browser) puts it in the project through a reviewed draft, as described above. The model lives in the project beside the detection rules, coefficients and all, so it travels with the plant and a refit shows in the diff — a refit moves an estimate with nothing else changing, and that is exactly the change worth reviewing. On the Desktop, **Put it in the project** does it directly: *'Case weight estimate' is in the project now. It travels with the plant, and a refit will show in the diff.*

Running a trained model at the edge

The Studio fits a linear model. A model trained anywhere else — a tree ensemble, a small neural network, an anomaly score — can run on the node as a **model signal**, as long as it is exported to ONNX. A model signal is a virtual signal whose value comes from the model: it is recorded, plotted, alarmed on and published like any computed signal.

Configuring a model signal

On the Desktop, **Acquisition ▸ Virtual Signals…**, **Add**, and choose the kind **Model**. The definition box holds the specification:

Field	Purpose
`model`	The model's name in the node's registry.
`features`	The model's input row, in order. Each names a `signal` and a `kind` : `last` , `lag` (the value `window` samples back), or `mean` , `min` , `max` , `stdDev` over the latest `window` samples.
`output` , `element`	Which output of the model, and which element of it, becomes the signal. Blank is the first numeric output, element 0.
`rateMs`	Run every so many milliseconds, or `0` whenever an input changes.
`staleAfterMs`	An input silent longer than this makes the output Uncertain; `0` does not check.
`budgetMs`	How long one inference may take. Default 100 ms.

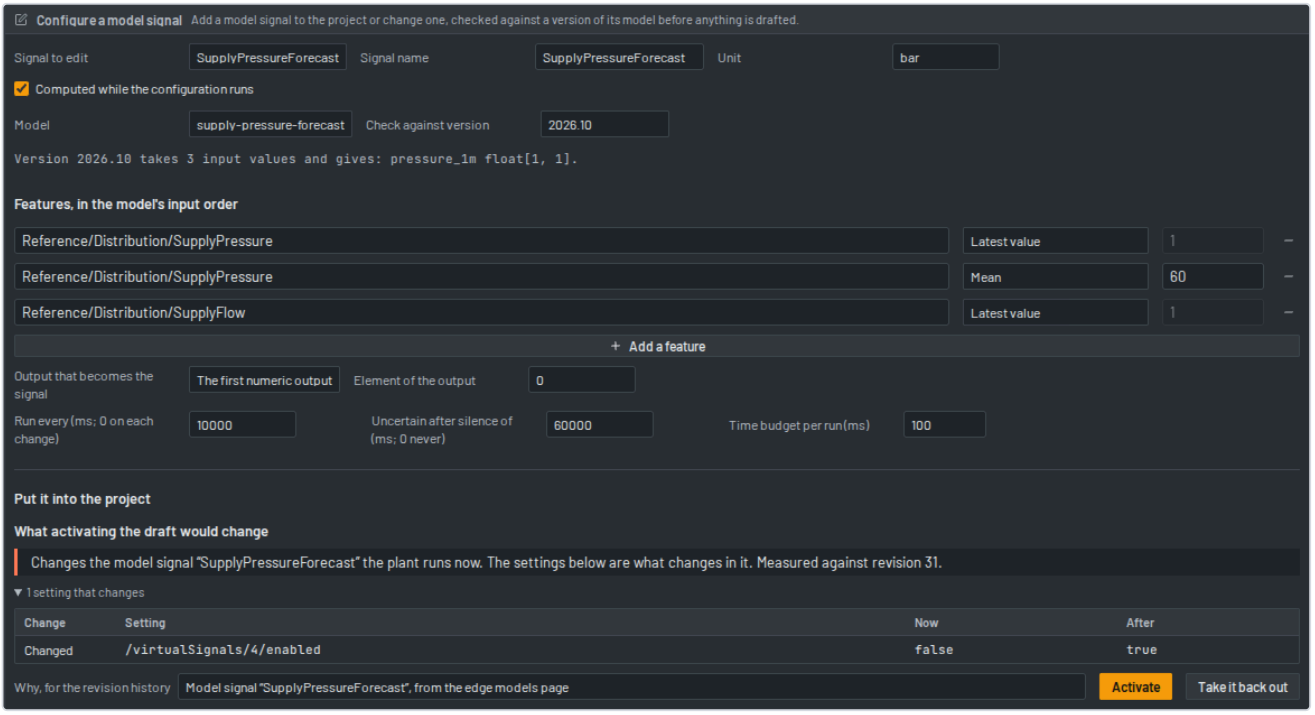
Windows count samples as they arrived, not time. Train the model on the same features.

In the browser, **Configure a model signal** at the foot of **Edge models** is the same specification as a form:

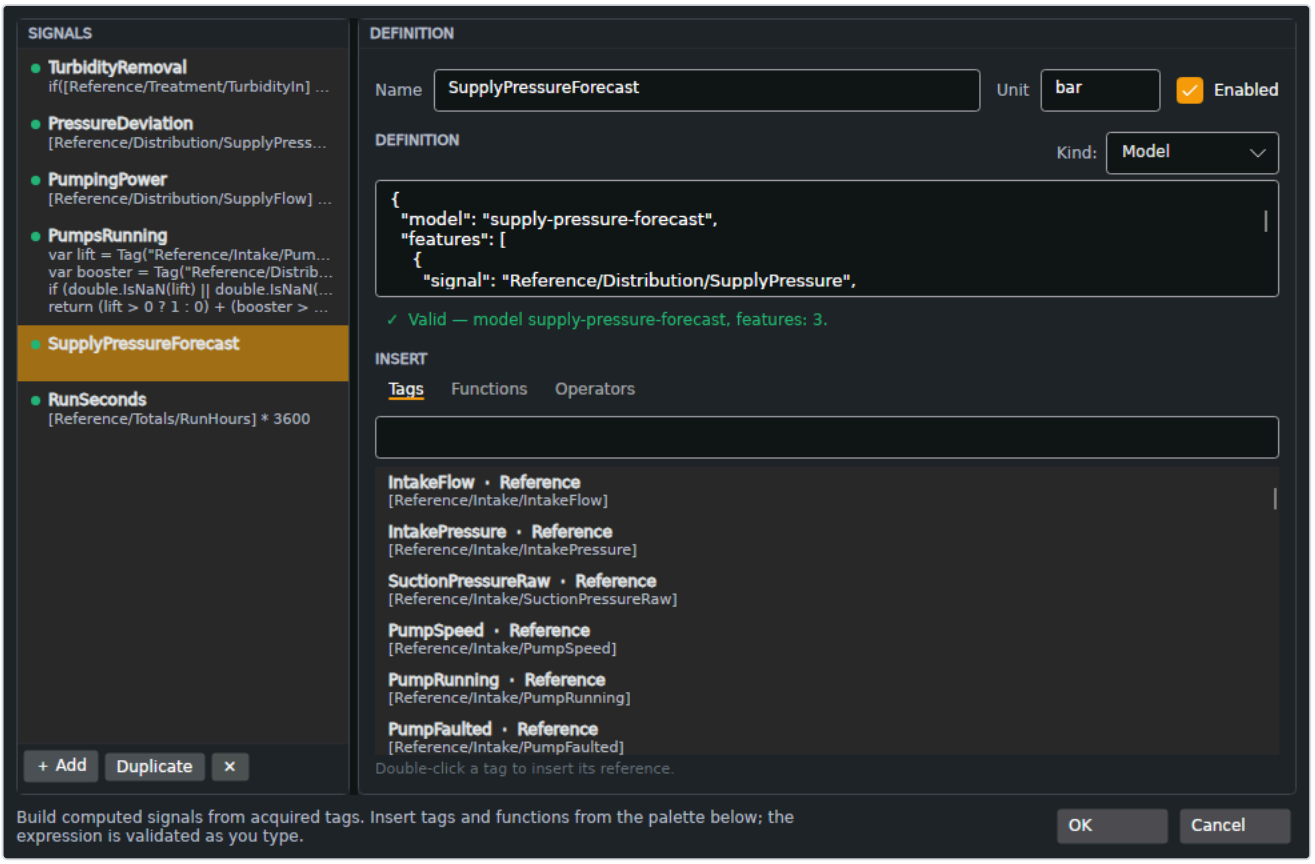
Element	Purpose
Signal to edit	A model signal the configuration holds, or *A new model signal*.
Signal name, **Unit**, **Computed while the configuration runs**	The signal's name, its unit, and whether it is enabled.
Model, **Check against version**	The model from the registry, and the version the form is checked against — the version deployed to the signal, or the newest.
Features	One row per input column, in order: the signal, the kind (latest value, value that many samples back, mean, minimum, maximum, standard deviation) and the samples it reads.

Element	Purpose
Output that becomes the signal, **Element of the output**	Chosen from the outputs the version declares; blank is the first numeric output.
Run every, **Uncertain after silence of**, **Time budget per run**	`rateMs`, `staleAfterMs` and `budgetMs`.
Not ready to be put into the project	Every reason the node would refuse it, as you type: a missing name, model or signal, a window, rate or budget out of range, a feature count that is not the model's input width, an output the version does not have or cannot turn into a number, an element outside it.
Put the model signal into the project…	Writes it into the node's draft and shows what activating would change. **Activate** puts it into service and then deploys the version it was checked against; **Take it back out** withdraws it from the draft.

The form is offered to someone the node lets configure signals and change configuration; anyone else reads why.



Configure a model signal with the reference works' SupplyPressureForecast opened, switched on and checked against version 2026.10, which takes three input values and gives pressure_1m: its three features (the latest supply pressure, the mean of its last 60 samples and the latest flow), a run every 10 s within a 100 ms budget, and the review — one setting changes, enabled false to true — ready to activate

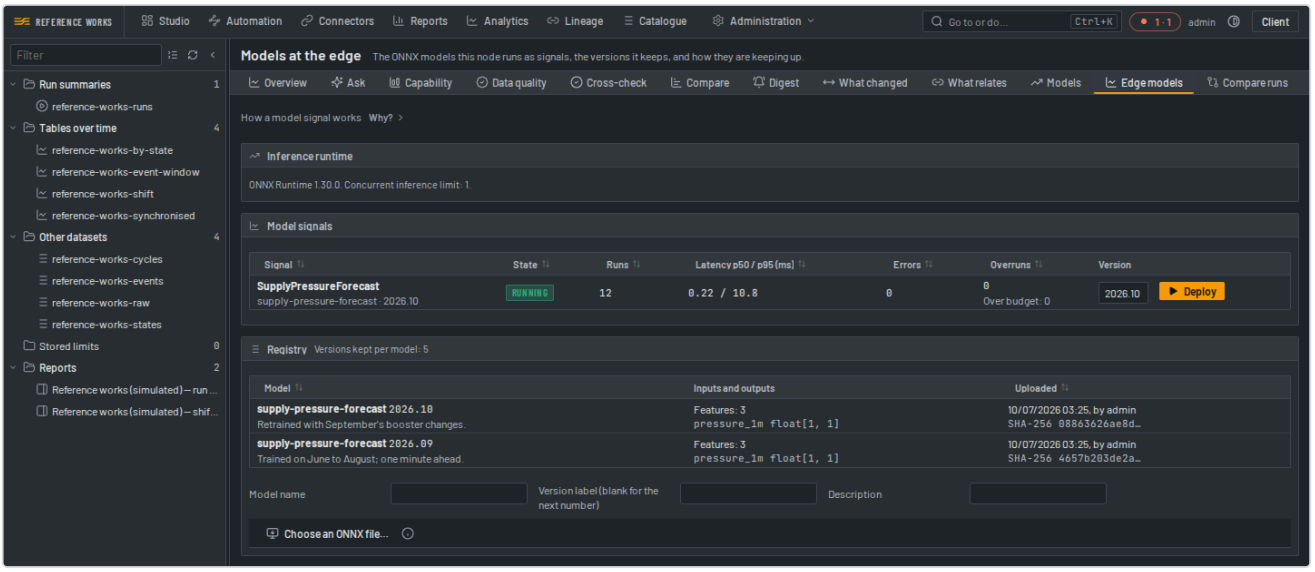


The same signal on the Desktop: Acquisition ▸ Virtual Signals with SupplyPressureForecast chosen, kind Model, its specification and "Valid — model supply-pressure-forecast, features: 3"

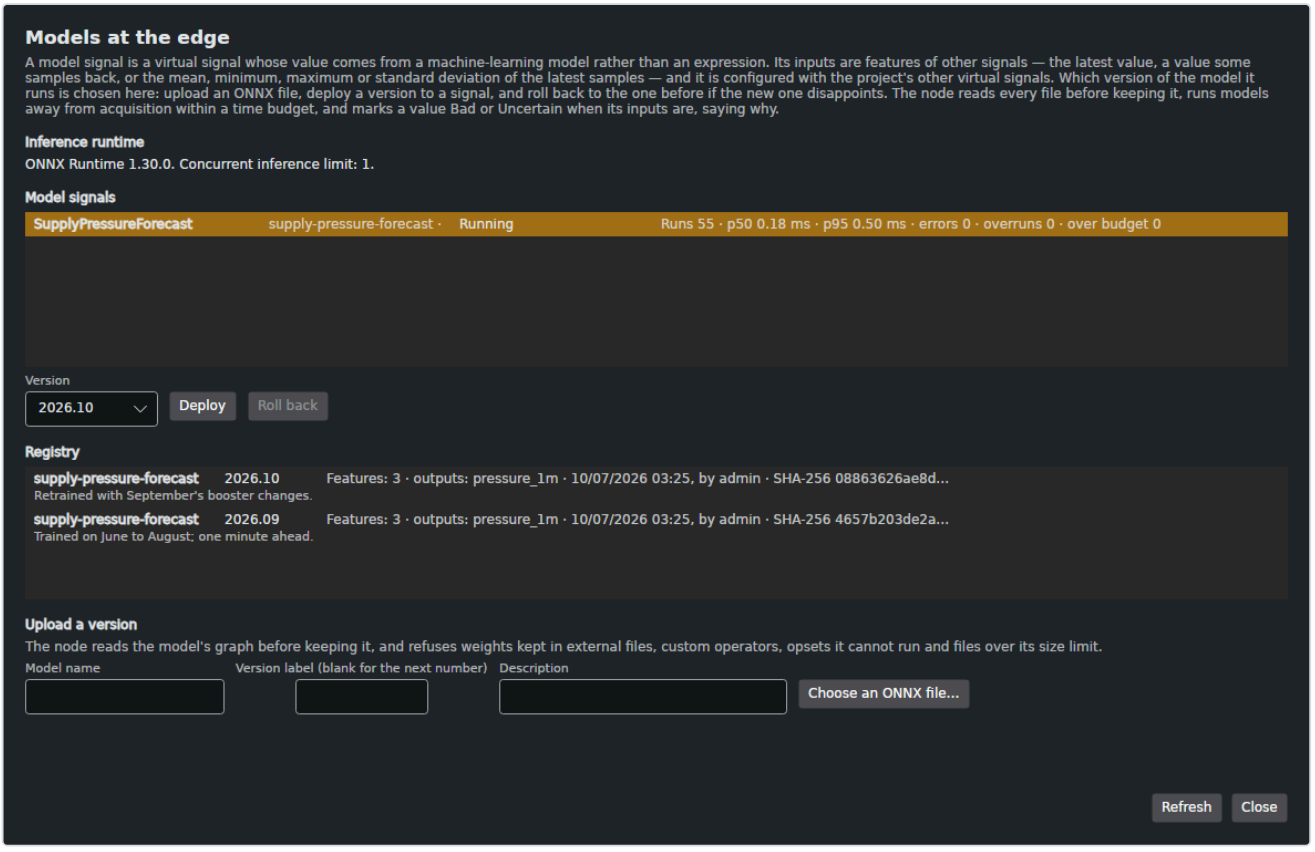
Uploading, deploying and rolling back

Data ▸ Models at the Edge… on the Desktop, **Edge models** beside **Models** in the browser. The screen reads top to bottom:

Element	Purpose
Inference runtime	The runtime installed on the node and how many inferences it runs at once, or why model signals cannot run.
Model signals	Each model signal's state, runs, latency p50 / p95, errors, overruns and runs over budget, and the node's own sentence when it is not running.
Deploy	Runs the chosen version on the selected signal. Refused when the version does not fit the signal as configured — a different number of features, or an output it does not have.
Roll back to …	Returns the signal to the version it ran before.
Registry	Every version kept: who uploaded it, when, its SHA-256 and its inputs and outputs.
Upload a version	Sends an `.onnx` file under a model name (the file's name if blank) and an optional version label and description.



Edge models in the browser: ONNX Runtime 1.30.0 installed; SupplyPressureForecast running version 2026.10 with its runs, latency, errors and overruns; and the registry keeping versions 2026.10 and 2026.09 with who uploaded them, their checksums and their inputs and outputs



Data ▸ Models at the Edge on the Desktop: the runtime, the forecast running with its runs and latency, the version to deploy, and the registry

Uploading, deploying and rolling back need the permission to configure signals. On the Desktop, a role without it sees the window with the buttons off and the reason in their place. Each is recorded in the audit trail before it happens. The node keeps the last five versions of a model and never removes one that is running or that a rollback would return to.

On a plant server, where the recorder service holds the node and the web host serves the browsers, the browser's **Edge models** page uploads, deploys and rolls back too: the web host asks the node as the person signed in, so the node checks their permission to configure signals, reads and verifies the file itself

(the web host only carries it, and refuses a file over the node's own limit before sending it), and records `models.upload`, `models.deploy` and `models.rollback` in its trail under their name, before and after. A bearer token sent to the web host names no person and is refused there; an integration publishes a model through the node's own data API. Who is offered the buttons there is whoever the node lets configure signals: the account every browser on a plant server reads with may change nothing, and is not what decides.

On a workstation joined to a node elsewhere, **Data ▸ Models at the Edge…** does the same: an upload, a deployment and a rollback are asked of the node as the person signed in at the workstation, not with the workstation's own data account, so the node checks their permission, reads the file itself and its trail names them. A node of a build from before this could be asked says that it needs updating.

What the node refuses

An uploaded model is a program written elsewhere, and the node reads it before keeping it. It refuses a file over 32 MB; weights kept in external files; custom operators (only the standard ONNX and ONNX-ML operators run); an opset outside 7 to 23; more than one input; an input whose feature count is not fixed; and outputs that are maps or sequences rather than numbers. Every reason is listed at once.

What a model signal's quality means

The model signal is	When
Good	The model ran within its budget on good inputs.
Uncertain	An input is Uncertain, or has been silent longer than `staleAfterMs`.
Bad	An input is Bad; a window is not yet full; nothing is deployed; the version does not fit; no runtime is installed; or the inference failed or ran past its budget. The reason travels with the value.

A model that cannot keep up never slows acquisition: a run that would start while the previous one is still going, or while the node is at its limit, is skipped and counted as an overrun.

The inference component

The ONNX runtime is installed separately, because its native library is larger than the rest of a node service. Copy the `inference` folder produced by `scripts/publish-inference.sh` (or `publish-inference.ps1`) beside the program that holds the node. Without it, model signals are Bad with *No inference runtime is installed on this node*.

No restart is needed. A node without the component looks at the folder every few seconds and loads it once the copy has finished; its model signals recover at their next run. A component that fails to load is reported under **Inference runtime** and is tried again only when its files change; acquisition carries on either way. Replacing a component that is already loaded needs a restart.

There is no component for Intel Macs (`osx-x64`). ONNX Runtime 1.30.0 has no x86_64 macOS build — its last one was 1.23.2 — and the script refuses that platform rather than ship an older runtime beside the current one. On an Intel Mac, model signals say that no runtime is installed.

Where an assistant fits

The arithmetic runs with no AI provider configured. Where the Studio Assistant is licensed and a model is configured for shortlist advice, each shortlist entry can carry a note — whether it looks **downstream** of the target, the **same measurement** as another row, **plausible**, or a **coincidence**. Only the signal paths

on the shortlist, the name of the target and each candidate's correlation and delay are shown to it; no readings. Its notes never change the order, the correlations or the lags, and a note about a signal not on the list is discarded. An unreachable provider produces no notes and no error.

Troubleshooting

Symptom	Cause and remedy
The page says the licence does not include the Models module.	The module is not licensed. Correlation and investigation remain available under **Analysis**.
Pick the signal to be estimated. / *'...' is not a signal this node has.*	Choose the target from the list.
That is not a period this understands.	Use forms such as `30d`, `12h`, `15m`.
Nothing moves with it over this period.	These signals do not carry the answer, or the delay is longer than 4 hours. Try a longer period or a wider **Search among**.
The licence permits N signals in one question.	Narrow **Search among**.
Estimate now is disabled.	The fit is not usable — the verdict says why. For a forecast, it did not beat carrying the last value forward.
Watch it is disabled.	Same gate as the estimate.
An estimate is flagged as an extrapolation.	An input is outside its fitted range. Refit over a period that includes this operating region.
Many measurements were left out.	Inputs had no samples for them. Check that the inputs were recorded over the whole period.
The watch reports measurements it could not judge.	An input left its fitted range; the watch was blind then. A wider fit fixes that; a lower threshold does not.
A replay is refused as too long.	The period holds more than 100,000 target measurements. Use the shorter period it names.
A model signal is Bad: *No inference runtime is installed on this node.*	Install the inference component beside the program that holds the node. It is picked up within a few seconds; no restart. On an Intel Mac there is no component.
Inference runtime says the component could not be loaded.	The folder is incomplete or for another platform. Copy it again; the node tries again when the files change.
The browser lists *Not ready to be put into the project*.	Correct what each line names. *The model takes N input values* means the features do not match the chosen version's input.
Configure a model signal is not shown.	The node has not granted you the permission to configure signals.
A model signal is Bad: *No version of … is deployed to …*	Deploy a version to it under **Models at the Edge**.
A deployment is refused: *The model takes 4 feature(s) and the signal is configured with 3.*	The model and the signal's features disagree. Change the specification, or deploy a version trained on these features.
An upload is refused for external files or custom operators.	Re-export with the weights inside the model and only standard operators.
Overruns keep rising.	The model is slower than its rate or input changes. Raise `rateMs`, simplify the model, or raise `budgetMs`.

The Studio Assistant

> **Studio Assistant (`studio-assistant`).** This chapter describes the Studio Assistant, an optional authoring aid inside the browser Studios and the Models Studio's shortlist. It is licensed separately, as a capability in the licence rather than as an entry under **Administration ▸ Modules**, and this chapter is present only on installations licensed for it. Both Studios, and the Models Studio, are fully functional without it.

What it does

The Studio Assistant is a second author inside the Studios. You describe a change in your own words; it answers with a **proposal** — a list of typed operations against the document you have open — and the Studio applies that proposal through the same commands its own panels use. An element the Assistant added is indistinguishable from one dragged off the palette: the same command created it, the same validator judges it, one undo entry reverses it, and the same **Save** stores it.

It is available in three browser Studios:

Studio	What it can help compose
HMI Studio	Operator pages: components, their placement, text, style, format, thresholds, bindings and navigation.
Report Studio	Reports: elements, tables, charts, data sources, parameters and page setup.
Automation Studio	Rules: blocks, their settings and wiring. The result is a draft; a person activates it.

On the Desktop, the same licence lets a model add notes to the Models Studio's shortlist (see *Shortlist advice on the Desktop* below). That is the only place the Desktop asks a model anything.

What it deliberately is **not**:

- **Not a document generator.** It has no operation that replaces a document.
- **Not an autonomous agent.** Nothing is applied without an author, except the low-risk changes an installation opts into.
- **Not a query tool.** It writes no SQL and runs no query you did not save as a dataset.
- **Not a way to command the plant.** It has no operation that writes a tag, moves a setpoint or acknowledges an alarm.

What must be true before it answers

Four things, checked in this order on every request:

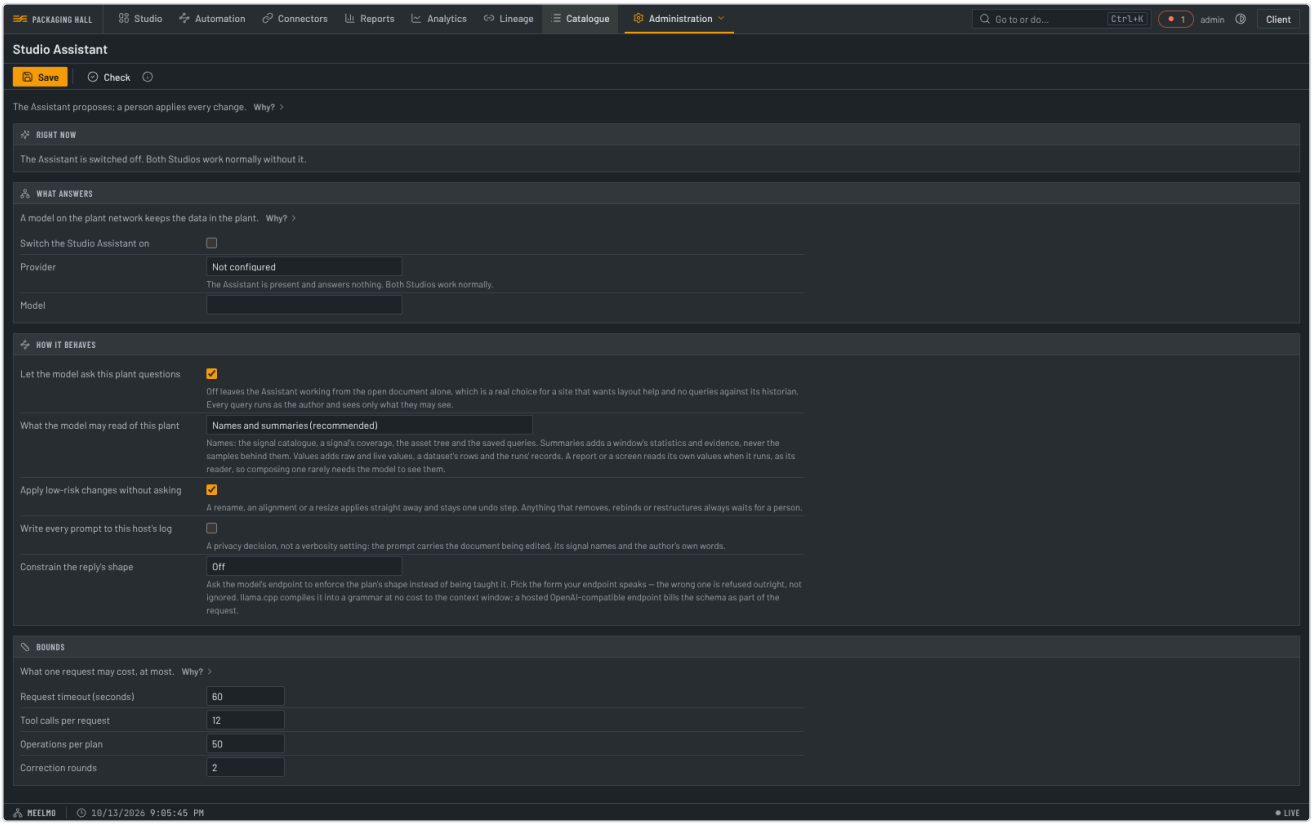
1. **Signed in.** The Assistant is used by a person through a browser session, never by a bearer token.
2. **Licensed.** The licence includes the Studio Assistant capability. It is independent of the MCP server capability: a site can have either without the other.
3. **Configured.** An administrator has switched it on and chosen a model provider that answers.
4. **Permitted.** You hold the authoring permission for that Studio.

Studio	Permission needed (for every mode, including Analyse)
HMI Studio	`Hmi.EditPages`
Report Studio	`Reports.Design`
Automation Studio	`Automation.Edit` (never `Automation.Activate`)

When the licence or the site's switch withholds it, the Studio looks exactly as it did before the Assistant existed: no **Assistant** tab and no **Ask** verb. When it is switched on but no model is configured or none is answering, the tab stays and says which, and an administrator is offered **Open Assistant settings**.

Settings in the browser

Administration ▸ Studio Assistant (`/admin/assistant`) needs the `Administer` permission.



Administration ▸ Studio Assistant with the Assistant switched off: Save and Check on the toolbar; the Right now panel; What answers with the on switch, Provider and Model; How it behaves with the plant-query switch, how much of the plant the model may read, auto-apply, prompt logging and reply-shape constraint; and Bounds with the request timeout, tool calls, operations per plan and correction rounds

Panel	Field	Default	Meaning
Toolbar	**Save**	—	Writes the configuration. The next question goes to the new model; no restart is needed. A question already in flight finishes against the model it started with.
Toolbar	**Check**	—	Asks the candidate model one throwaway question — no document, no catalogue, no tools — and reports what came back and how long it took. Saving does not connect, so check before you save.

Panel	Field	Default	Meaning
Right now	—	—	What is true at this moment: whether the licence covers the Assistant, whether a model is reachable, and any problem the host found at start-up. For example *The Assistant is switched off. Both Studios work normally without it.*
What answers	**Switch the Studio Assistant on**	Off	Off by default, and stays off after an upgrade.
	Provider	**Not configured**	**Not configured** (present, answers nothing); **Anthropic — Claude** (the vendor's hosted API; its address is fixed and the document being edited leaves the plant); **OpenAI-compatible endpoint** (anything serving `/v1/chat/completions` — llama.cpp, vLLM, Ollama, LM Studio, or a gateway of the site's own); **Built-in test provider** (answers from a fixed script, to prove the panel, permissions and audit trail before a model exists).
	Endpoint	—	For the OpenAI-compatible provider: the model's address. The server's base address, the `/v1` root or the full `/v1/chat/completions` route are all accepted.
	Model	—	Which model at that endpoint.
	API key	—	Sealed against this installation when saved, never sent back to the screen. An empty box showing *stored — leave blank to keep it* keeps the stored key. **Remove the stored credential** clears it. A local endpoint often needs no key.
How it behaves	**Let the model ask this plant questions**	On	Off leaves the Assistant working from the open document alone. Every query runs as the author and sees only what the author may see.
	What the model may read of this plant	**Names and summaries (recommended)**	**The names of what exists** — the signal catalogue, a signal's coverage, the asset tree, saved queries. **Names and summaries** adds a window's statistics and evidence, never the samples behind them. **Values too: samples, live values, dataset rows** adds raw and live values, a dataset's rows and the runs' records.
	Apply low-risk changes without asking	On	A rename, an alignment or a resize applies at once and stays one undo step. Anything that adds, removes, rebinds or restructures always waits for a person.
	Write every prompt to this host's log	Off	A privacy decision, not a verbosity setting: the prompt carries the document being edited, its signal names and the author's own words.
	Constrain the reply's shape	**Off**	**llama.cpp** or **OpenAI** asks the endpoint to enforce the plan's JSON shape. Pick the form your endpoint speaks; the wrong one is refused outright.
Bounds	**Request timeout (seconds)**	`60`	5–600. The default suits a model on the plant network; a hosted endpoint composing a rule has been measured at 58–85 seconds, so raise it for one.
	Tool calls per request	`12`	0–64. Plant questions one request may ask, across its correction rounds.

Panel	Field	Default	Meaning
	Operations per plan	`50`	1–200. No setting can exceed 200.
	Correction rounds	`2`	0–5. How often a refused plan may be sent back to the model with its faults.

Fields set through `MEELMO_ASSISTANT_*` environment variables keep winning over the file; the screen lists them under *These fields come from the environment…* rather than reporting a save that changed nothing. If the licence does not include the Assistant, the screen says so instead of letting anybody configure a provider that will never answer.

Where the setting is stored

The screen writes the `assistant` section of the web host's `web-config.json` in its state directory:

Install	File
Windows service	`C:\ProgramData\Meelmo\web\web-config.json`
Linux service	`/var/lib/meelmo-web/web-config.json`
Container	`/var/lib/meelmo/web-config.json`

Editing the file by hand is equivalent, but **add the section; never replace the file** — the same file holds the listener, its certificate and which node the host serves. A hand edit takes effect when the host restarts.

A minimal section for a model on the plant's own hardware:

```
"assistant": {
  "enabled": true,
  "provider": "local",
  "endpoint": "http://inference.plant.local:8000/v1",
  "model": "<the model name your inference server serves>"
}
```

Worked example: switching it on against a model in the plant

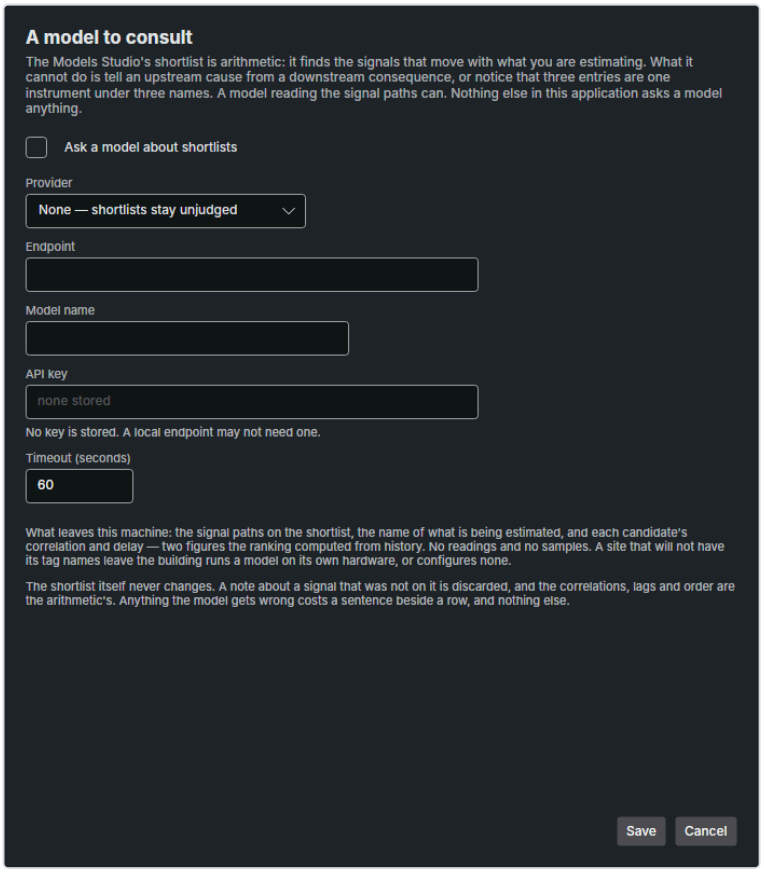
The site runs an inference server at `http://gpu-01.plant.local:8000` on the plant network.

1. Sign in to the browser as `admin` and open **Administration ▸ Studio Assistant**.
2. Tick **Switch the Studio Assistant on**.
3. **Provider: OpenAI-compatible endpoint**. **Endpoint:** `http://gpu-01.plant.local:8000/v1`. **Model:** the name the server reports. Leave **API key** empty.
4. Leave **Let the model ask this plant questions** ticked and **What the model may read of this plant** at **Names and summaries (recommended)**.
5. Leave **Apply low-risk changes without asking** ticked.
6. Press **Check**. **Right now** reports what answered and how long it took.
7. Press **Save**. The banner reads *Saved. The next question goes to this model; no restart is needed.*

Because the endpoint is on the plant network, nothing the Assistant sends leaves the site.

Shortlist advice on the Desktop

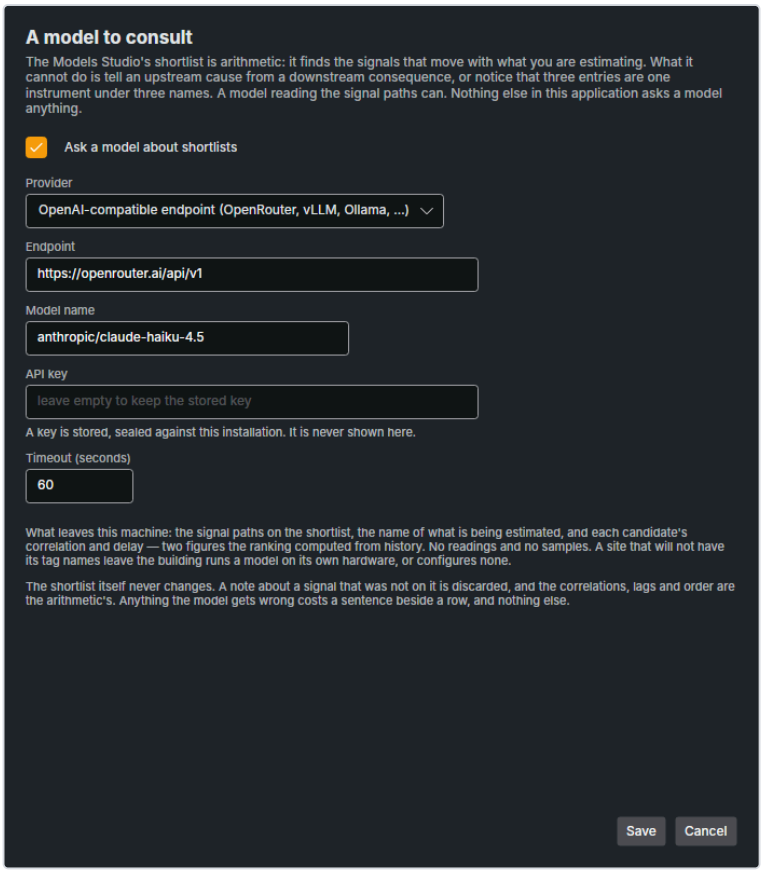
Data ▸ Model for shortlist advice… opens *A model to consult*. It configures the model the Desktop's Models Studio may ask about a shortlist — which entries look downstream of the target, and which are one instrument under several names. It does not affect the browser Studios, which use the settings above.



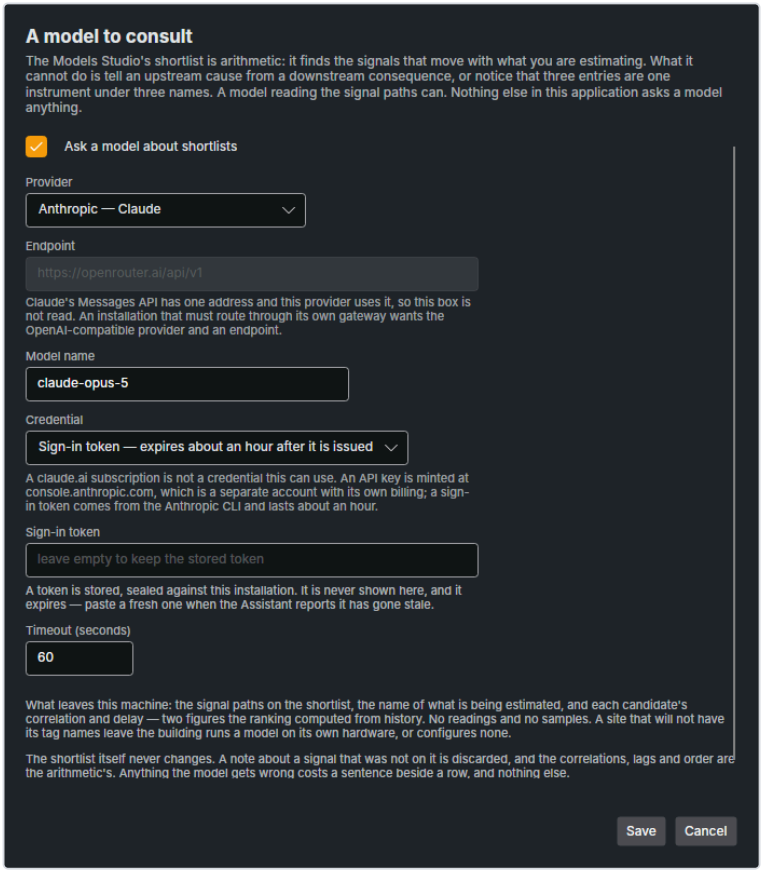
A model to consult, as first opened: Ask a model about shortlists unticked, Provider set to None — shortlists stay unjudged, no key stored, and the statement of exactly what would leave this machine

Field	Meaning
Ask a model about shortlists	Off by default.
Provider	**None — shortlists stay unjudged**, **Anthropic — Claude**, or **OpenAI-compatible endpoint (OpenRouter, vLLM, Ollama, …)**.
Endpoint	For the OpenAI-compatible provider. Not read for the Anthropic provider, whose address is fixed.
Model name	Which model to ask.
Credential	Anthropic provider only: **API key** (good until revoked) or **Sign-in token** (expires about an hour after it is issued).
API key / **Sign-in token**	Sealed against this installation and never shown again. *leave empty to keep the stored key* keeps it. A local endpoint may need none.
Timeout (seconds)	Default `60`, between 1 and 600.
Save / **Cancel**	*Saved. The next shortlist will be shown to it.*

The window says exactly what leaves the machine: the signal paths on the shortlist, the name of what is being estimated, and each candidate's correlation and delay. No readings and no samples.



A model to consult configured for an OpenAI-compatible endpoint: an endpoint address, a model name, and a stored API key shown only as leave empty to keep the stored key



A model to consult with the Anthropic — Claude provider chosen: the Endpoint box is not read, a model name, and Credential set to Sign-in token with a note that the token expires about an hour after it is issued

The shortlist itself never changes. A note about a signal that was not on the list is discarded; the correlations, delays and order are the arithmetic's. A provider that is unreachable or answers with something unreadable produces no notes and no error.

Asking it in the HMI Studio

Where it is

The Assistant is a tab in the HMI Studio's right rail, beside **Properties**, and an **Assistant** verb in the Studio's row of verbs. The tab and the verb are absent — not disabled — when the site has no Assistant, which is why neither appears here:



The HMI Studio editing the packaging line's overview page on an installation where the Assistant is switched off: the right rail shows only Properties, and there is no Assistant verb in the toolbar

In the Report Studio the tab sits beside **Properties**, **Settings** and **Problems**, with **Ask Assistant** in the editing toolbar. In the Automation Studio it sits beside **Properties**, **Test run** and **Replay**.

The panel

Element	Meaning
Context chips	What the Assistant is being shown: *Page: overview*, *Selected: …* or *Selection: …*, *Validation findings: N*, and *Read-only* when the page cannot be changed.
Mode	**Build** (create what does not exist), **Modify** (change the page or the selection), **Analyse** (explain or critique; never proposes a change), **Fix** (correct what the validator reports, minimally).
Starter prompts	On an empty conversation, for example *Build an overview screen for one machine*, *Add a trend of the signals I have selected*, *Explain this page and where its controls lead*. Pick one and edit the wording before sending.

Element	Meaning
Text box	*Describe the change you want, or ask a question about this document.* **Enter** sends, **Shift+Enter** starts a new line, **Escape** stops.
Send / **Stop**	Send the request; stop waiting for it.

The mode is authority, not phrasing. An **Analyse** request is never read as a plan, whatever the model writes back.

Constraints you state are enforced. Phrases such as *without changing the bindings*, *don't delete anything*, *keep the current scope* and *only these* are turned into rules the plan is checked against; a plan that breaks one is refused, not negotiated.

The proposal

A plan comes back as **Proposed change**:

Element	Meaning
Summary	One line to read before deciding.
Risk	**Low risk**, **Medium risk** or **High risk**, computed by the product from the operations — never reported by the model.
Read this before applying it.	Shown when review is required.
Change list	One line per operation with `+` added, `~` modified, `−` removed, its target and reason. Under each change to something already there, **Each value, before and after**: for example `Title: Line rate becomes Line rate (cases/min)`.
Looked up:	The plant questions the model asked.
Apply / **Discard**	Apply the whole plan as one undoable step, or throw it away.
Review details	The document, its revision, the plan id, the operation count and each **Operation payload**, for whoever is diagnosing a plan.

After **Apply**, the changed components are selected, the change list stays on screen, and the undo history has one entry named *Assistant: <summary>*. **Ctrl+Z** reverses the whole plan in one step; redo puts it back.

Worked example: adding a trend to the packaging overview

The packaging line's `overview` page has a **Rate and OEE** trend. The shift engineer wants a second trend of case weight against its target.

1. Open the `overview` page in the HMI Studio and click the **Assistant** tab.
2. **Mode: Build**.
3. Type: *Add a trend of Packaging/Checkweigher/CaseWeight and Packaging/Checkweigher/WeightTarget below the Rate and OEE trend, titled "Case weight". Don't change anything else.*
4. Press **Enter**. The panel shows *Asking the Assistant…*; the model looks the two signals up in the catalogue (they appear under **Looked up:**).

5. The proposal reads, for example: `+ Add a trend "Case weight"` and `~ Bind the trend's series to Packaging/Checkweigher/CaseWeight and Packaging/Checkweigher/WeightTarget`, at **Medium risk** (it adds a component and binds it).
6. Read the change list and press **Apply**. The new trend is selected on the canvas.
7. Look at it in **Preview**. If it is not what you wanted, **Ctrl+Z** removes both operations at once.
8. Press **Save draft**, then **Publish…** as usual — the Assistant never saves.

To fix validation problems instead: press **Validate** first, so the findings are current, then choose **Fix** and ask *Fix the problems the validator is reporting.* The HMI Studio validates on the server when you press **Validate**, save or publish, and **Fix** works from the last validation.

What it can and cannot change

It can

Studio	Operations
HMI Studio (17)	Add or remove a component; place it on the grid or canvas; set a declared property; heading, subtitle, description, name; style (only the fields named change); number format; thresholds; bind one slot or a trend's list of signals; align, distribute, order, lock; navigation of the **observational** kinds only; page settings (name, description, type, grid, canvas).
Report Studio (25)	Add, remove, move, rename, style elements; set a declared property; bindings; visibility conditions; conditional formatting; configure tables, charts and repeaters; align, distribute, order; add or remove pages; page settings and band heights; data sources and parameters; the report's scope, title and description.
Automation Studio (9)	Add, remove, move, rename blocks; set a declared setting; wire and unwire; switch a block off or on; the flow's name, description, enabled flag and trace detail.

Every one of these maps onto a command a person can also perform in the Studio.

It cannot

- **Command the plant.** On an HMI page the only commanding constructs are a *write tag* or *acknowledge alarm* interaction. The Assistant's navigation operation accepts only the thirteen observational kinds (none, navigate to page or asset, open trend, alarm or execution details, select asset, area or time range, set page parameter, open information, card or dialog). A plan asking for a commanding kind is refused three times over: by the host when reading it, and in the browser before and during applying it. Authoring a control is a person's act.
- **Save, publish or activate.** Committing is **Save**, **Publish** or activation, with the same permission, validation and audit as always. In the Automation Studio it cannot activate a revision, run or test a flow, install a package, edit a connector or write a secret.
- **Change who may see a page**, the draft/publish lifecycle or the edit lease.
- **Rename a report, change its schedule or its delivery.**
- **Replace a document, write a property by path, write SQL** or an expression outside the report expression language.
- **Bind a signal you may not read.** A plan naming one is refused, and the faults name each signal.
- **Edit a read-only document.** A report opened read-only, or a page whose edit lease is held by someone else, refuses every plan, and the panel switches to **Analyse**.

Approvals, risk and auto-apply

Risk is computed from the operations:

The plan	Risk
Removes anything, however little	**High**
Carries more than 8 operations	**High**
Rebinds more than 4 things	**High**
Changes the frame: scope, page setup, a band height, a new page, page settings	**High**
Clears something (thresholds, rules, a binding, an interaction, a setting value)	**High**
Rebinds anything, or changes a document setting	Medium
Adds an element or component	Medium
Geometry, style, text, order, locking	Low

With **Apply low-risk changes without asking** on, plans that need no review are applied at once and announced as *Low-risk changes are applied here without review.* That is Low only: a Medium plan (a rebinding, or anything added — a component, an element, an automation block) and a High plan always wait for a person, and so does a plan whose review found that it would silently drop existing content (for example a table reconfiguration that removes columns). The model may ask for review; it cannot clear review a plan deserves.

Stale plans are refused, not merged. If the document changed after you asked — your own edit, another tab, or a plan you applied — the proposal is refused with *The document has changed since this was proposed. Ask again.* The check is made in the panel and again inside the apply.

Only errors the plan introduced refuse it. A plan is staged, the Studio's own validator runs on the result, and if the plan introduced an error the whole thing is rolled back with the validator's words. Warnings do not refuse a plan, nor do problems the page already had, nor an unbound component the plan just added.

Privacy: what is sent, and where

Where it goes. Text you type, a snapshot of the open document, and the names the tools return are sent to **the model endpoint your site configured, and nowhere else**. Point it at a machine on the plant network and nothing crosses the site boundary. Choosing the hosted **Anthropic — Claude** provider, or a hosted gateway, means the document being edited — its structure, element and signal names and the validator's findings — leaves the plant on every question.

What the document snapshot contains. For an HMI page: its id and revision, its type and layout, the selection, the components around it with their placements and bindings, the validator's findings, and the component catalogue. Not the whole project, and not enough to reconstruct the document.

What the model may read of the plant is the **What the model may read of this plant** setting, enforced in the product: names only, names and summaries, or values too. Every query runs as the author, with the author's permissions and signal scope; an assistant cannot see further than the person driving it. What the data layer says about completeness — gaps, truncation, refused signals — travels with the answer to the model and to the author.

The credential is sealed on the host, never logged and never sent to the browser.

Prompt logging is off by default.

Nothing else. No telemetry, no usage reporting, no runtime downloads. The only outbound request the Assistant makes is the one to the configured endpoint.

What the audit trail records

Every proposal and every refusal is audited (`studio.assistant.hmi.plan`, `studio.assistant.report.plan`, `studio.assistant.automation.plan`) with the plan id, the mode, the document, its revision, the risk, whether review was required, and the kinds of operation. What became of each plan is recorded as `.applied`, `.discarded` or `.refused`, under the person who did it, including when the site's auto-apply applied it.

The trail deliberately does **not** record the prompt, the model's prose, the document's contents, any process value a tool read, or the API key.

Limits

Bound	Value
Operations in one plan	50 by default; never more than 200
Plant questions per request	12 by default, across all correction rounds
Correction rounds	2 by default
Request timeout	60 s by default
Conversation turns sent with a request	the last 12; the last 8 reach the model
One tool answer	6,000 characters (rows dropped, never characters)

Every plant question is also bounded by the data layer's own limits, with no exemption for the Assistant.

Troubleshooting

Symptom	Cause and remedy
No **Assistant** tab in the Studio.	You lack the Studio's authoring permission, or the licence does not include the Assistant, or the site has not switched it on.
The Studio Assistant is switched on and no model provider is configured.	Choose a provider, endpoint and model in **Administration ▸ Studio Assistant**. The host's start-up log names the missing field.
The Assistant's model provider is not answering.	Unreachable, refused, or slower than the timeout. Press **Check** on the settings screen; raise **Request timeout (seconds)** for a hosted endpoint. Manual editing is unaffected.
The endpoint answers 401 to a key that is plainly configured.	The key was sealed against another installation. Retype it on this host and save.
The stored credential is not sealed.	Retype it once and save to seal it.
That plan was refused, so nothing was changed.	The model could not produce a plan this build accepts. **Review details** lists each fault.
The document has changed since this was proposed. Ask again.	By design; ask again.
Fix did not fix what the Problems panel shows.	The page's findings are from the last validation. Press **Validate**, then ask again.
It is reasoning from this document alone; it cannot look anything up in the plant.	**Let the model ask this plant questions** is off, or you lack `ReadCatalog`.

Symptom	Cause and remedy
You are not permitted to make that change.	The plan named a signal you may not read.
It will not put a write or acknowledge control on a page.	By design, permanently.
Shortlists in the Desktop's Models Studio carry no notes.	**Ask a model about shortlists** is off, the provider did not answer, or the licence does not include the Assistant. The shortlist is unaffected either way.

Troubleshooting

This chapter is organised by **what you saw**, not by which part of the product produced it: a symptom, its usual cause, and what to do. Where Meelmo refuses to do something, it says why in a sentence; those sentences are quoted here as they appear, so you can search this chapter for the words on your screen.

Meelmo refuses rather than degrades. Most entries below are not faults but a deliberate refusal with the remedy in it. A host that would have put session cookies on an unencrypted network stops instead; a chart with a hole in its history draws the hole rather than a straight line; a live value whose stream has stopped says so rather than showing the last value it had. When something is missing from a screen, the first question is what the product decided not to pretend.

Some entries concern parts of the product that are optional modules. They are marked "if your installation includes the … module"; if yours does not, skip them.

Where to look first

Source	Where	What it tells you
The status bar	Bottom of the Desktop window	Mode (**Live** or **Playback**), recording (`REC:`), active alarms, and the historian's state. A historian problem stays on the status bar until it is no longer true.
The **Event Log** panel	**View ▸ Event Log**	What the application did: acquisition starting and stopping, connections lost and regained with the driver's own reason, configuration loaded or changed.
Administration ▸ Diagnostics…	Desktop	Live acquisition health per connection and group.
Administration ▸ Commissioning report…	Desktop	The node's own account of itself, check by check.
The application log	`<state>\logs\meelmo-YYYYMMDD.log` (**Help ▸ Open Logs Folder**)	Detail behind anything the screens summarise.
A host's console output	The service's log or console, from start	What the host is attached to, what it is serving and what it refused.

The state directory (`<state>`) is the installation's data folder: `C:\Meelmo\Data` in a standard Windows installation, or wherever `MEELMO_STATE` points. See the *Diagnostics and support* chapter for everything that is kept there.

It will not start

What you see	Why	What to do
A service starts, then stops, naming a bind address: *This host is bound to ..., which is reachable from the network, and no TLS certificate is configured.*	A routable address without TLS would put session cookies and tokens on the wire. This refusal is deliberate.	Configure a certificate (`MEELMO_WEB_CERT` , with its password's secret in `MEELMO_WEB_CERT_SECRET`), or bind to loopback. Only on a test bench, `--allow-insecure` overrides it and says so in the log every time.
The TLS certificate password could not be read from the secret store.	The certificate is configured, but its secret is not readable by the account the host runs as. Secrets are sealed per user.	Re-seal the secret as the service account, not as yourself.
"Unknown publisher" on the Windows elevation (UAC) prompt	The package was not signed.	Do not continue. Return it to your supplier.
A host serves nothing on the address you expect	It started on loopback, which is where it starts by default.	Add TLS and re-register the service with a routable bind.
The node at ... could not be started.	Usually another process already holds the node.	Close the other holder; the message names it.
This node is already being served by another Meelmo Agent (pid=... machine=... ...).	One node has exactly one Agent. The Desktop, the recorder service or a web host already has it.	Attach to the one that has it instead of opening the node a second time. **Do not delete `node.lock`** : the hold belongs to the running process, not the file, and deleting it would let two hosts poll the plant and write one historian. A crashed host leaves no hold, so there is never a stale lock to clear.
The node directory '...' is on a nfs4 filesystem, where an exclusive hold does not exclude another machine.	Network filesystems (NFS, SMB) cannot guarantee that only one machine holds the node. Retrying will not help.	Put the node directory, and the historian's files, on storage local to the machine. A shared volume is safe only when a cluster manager mounts it on one machine at a time.
Another Agent has taken this node... it has stopped acquiring.	Two Agents held the node directory at once, and this one stood down.	Find what took the node (usually a backup restore or a file sync writing into the node directory), then restart this host once the directory has one Agent.
No license installed and the trial could not be started.	The trial marker could not be written to the state directory.	Make the state directory (`MEELMO_STATE`) writable by the account the program runs as.
Audit log verification FAILED — the chain is broken in: ... at start-up	One day's audit file no longer follows its hash chain. It is shown at every start until that file is gone.	Read the message: *a writer began a new chain there* usually means a write cut off by a power loss or a locked file, not an edit; *names a different entry before it* or *was changed after it was written* should be investigated as a possible edit. Keep the file; do not edit it to "fix" it. See the *Users, roles and security* chapter.

Nobody can sign in

What you see	Why	What to do
Invalid username or password.	The name or password is wrong.	Check both; passwords are case-sensitive. An administrator can reset a password in **Administration ▸ Users & Security…**.
Too many failed attempts. Try again in … s.	The account is temporarily locked after repeated failures.	Wait for the time shown.
Sign-in is not enabled for this installation.	The installation was set up without accounts.	Complete the first-run setup, or ask the administrator who commissioned it.
If your installation includes the Web HMI module: *No web users exist and this host has no node to ask, so nobody can sign in. Open /setup in a browser…*	The browser host has not been configured.	Open `/setup` from a browser on the host itself (no key is needed there). From another machine it asks for the setup key printed on the host's console.
If your installation includes the Web HMI module: *No web profiles exist yet. Anyone this node knows can sign in…*	Not an error: the node is the account authority.	Sign in with a node account (one created in the Desktop).
If your installation includes the Web HMI module: *That node is already in service, and it did not accept that name and password.*	You are attaching to a node that someone has already commissioned.	Use an administrator that node already knows, with their password.

It will not connect

What you see	Why	What to do
A controller connection will not come up	The address or port is wrong, the controller is not reachable from this machine, or a firewall blocks it. The product does not retry silently.	Read the driver's own reason in the **Event Log**. Check the address and port from this machine. Use **Test Connection** in **Acquisition ▸ I/O Configuration…**.
The Desktop cannot attach to the plant server	**Administration ▸ Deployment…** defaults the node control port to `8443`, but a recorder serves the control plane on **8444**.	Change the port on that screen. This is the most common commissioning mistake.
Attached to the node at … but not to its live acquisition at …:…. History and the HMI are served; real-time endpoints will refuse.	The node's control plane is reachable but its live acquisition carrier is not. The carrier is a separate connection on its own port, **9701** by default.	Open port 9701 between the machines, or correct the server address and port in **Administration ▸ Deployment…**.
A connection works from a laptop but not from the server	Network segmentation, or a firewall on 9701, 8443 or 8444.	Check the ports: 9701 (acquisition carrier), 8443 (browser, if your installation includes the Web HMI module), 8444 (node control plane).

What you see	Why	What to do
A driver connects, but the password seems to be wrong	The configuration was copied from another machine. Secrets are sealed with a key bound to the machine and do not travel.	Re-enter the password on this machine.
An OPC UA server refuses the credentials	The driver will not send a password in clear text unless told to.	Use an encrypted endpoint, or set the override deliberately.
The OPC UA server's certificate '…' (thumbprint …) is not trusted … To trust it, move … into …/opcua/client-pki/trusted.	A secured OPC UA connection met a server certificate that neither the site PKI nor the client's trust folder vouches for.	Check the thumbprint with the server's administrator, then move the named file from `opcua/client-pki/rejected` to `opcua/client-pki/trusted`. The next attempt connects.
An OPC UA connection on Sign or Sign and encrypt is refused with *BadCertificateUntrusted* by the server	The server does not trust the client's own certificate yet.	On the server, move the client's certificate from its rejected folder to trusted (on a Meelmo node, `opcua/pki/rejected` to `opcua/pki/trusted`). The log of the reading node names the client's certificate file.
All … licensed client seats are in use. Ask one of them to disconnect, or add seats to this installation's license.	Client seats are concurrent, not named, and all are taken.	The message lists who holds them. Ask one to disconnect, or license more seats.
A Professional license is required to serve clients. (also for headless recording and for running an acquisition server)	An edition limit, not a fault.	Install a licence that includes it (**Help ▸ License…**).
If your installation includes the Web HMI module: the browser says *Not secure*, and live values stay stale behind **Live stream down**	The browser does not trust the site's certificate. Clicking through the warning loads the page but not its live connection.	Trust the site's certificate (the sign-in page has a link for it).

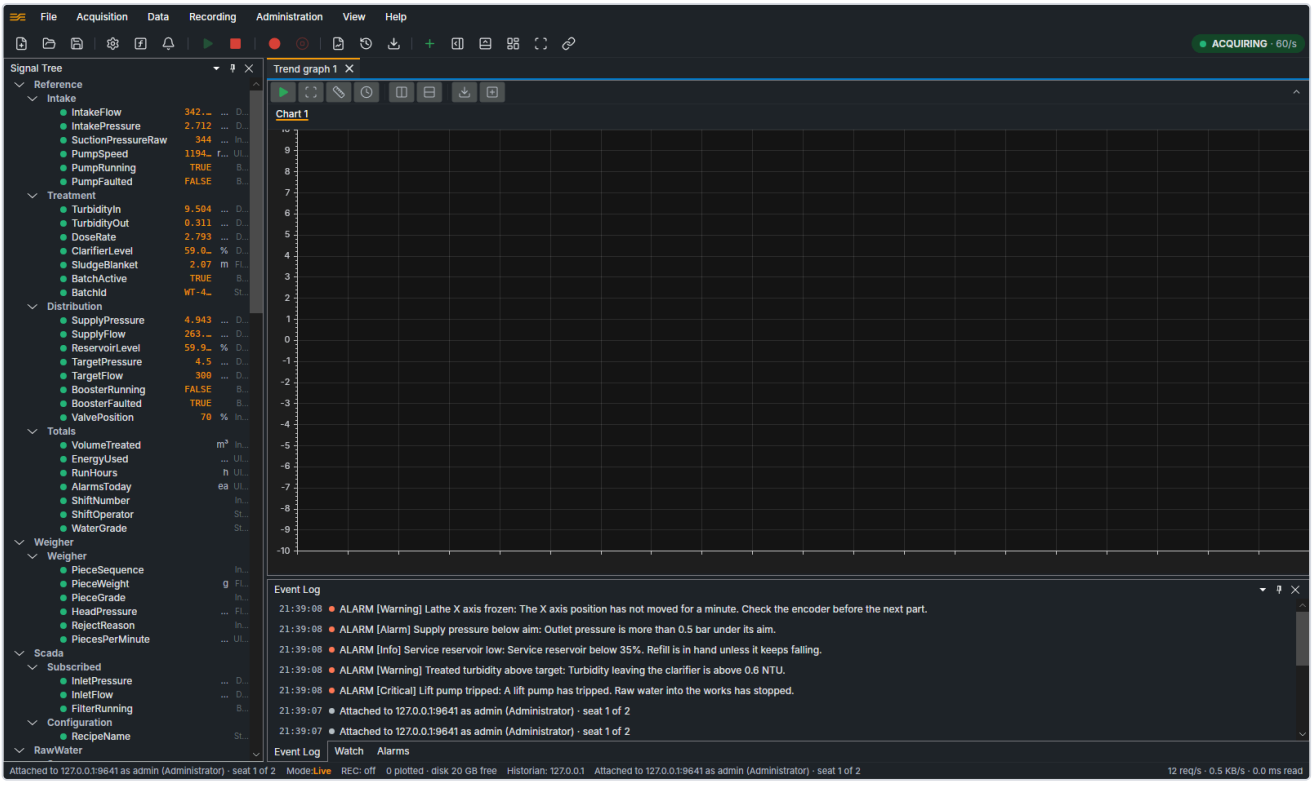
The data is not what you expected

What you see	Why	What to do
A value moves beside the signal, but its trend is empty	The value is live and the history is not: nothing has been recorded for it yet, or no historian is configured and the node does not record continuously.	Check the historian in **Data ▸ Historian…** (see *The historian is unavailable* below). A node with no historian serves its continuous recordings as history, so turning continuous recording on is the other remedy.
A trend draws only the last few seconds on a new installation	The historian has nothing older yet.	Nothing to fix; history accumulates from now.
A trend has a hole (a gap band) in it	A gap is shown, never interpolated: acquisition stopped (the node or its host was stopped or crashed, or the connection was disabled or restarted), or the historian was down. A missing value is missing, never zero.	Check **Administration ▸ Diagnostics…** for sequence gaps, and the historian's spool for an outage.

What you see	Why	What to do
The actual period does not match the requested one	Polling is best effort on a general-purpose operating system.	Open **Administration ▸ Diagnostics…** and compare **req** and **act**. If the timing must be defensible, use a source-synchronous acquisition mode.
The acquisition pill at the right of the toolbar shows `⚠ … overruns`	A group's poll rate is faster than the controller can answer.	**Administration ▸ Diagnostics…** shows which group (**ovr**). Slow that group down or split it.
Samples are being dropped	Best-effort recording under load, or a subscriber that stopped reading.	Switch the recording to **Lossless**, or reduce what is subscribed. The Recorded/dropped counters in Diagnostics say which.
Two signals do not line up in time	They are in different acquisition groups, so they do not share a timestamp.	Put them in one group. **Recording ▸ Recording Timing Analysis…** shows what the timestamps are worth.
Every tile or value goes stale at once	The live subscription has dropped, not the plant.	Check the connection indicators on the status bar and the **Event Log**.
One value is stale while its neighbours move	Often correct: a value subscribed "on change" is delivered only when it changes; a boolean true for an hour is still true.	If it is marked stale, its stream really has stopped, or the reading really is old. Check the driver.
Every value from one MQTT connection turns Uncertain together	The driver's stale-data watchdog: nothing arrived on that connection for longer than its **Stale after (ms)** setting (60 s by default). It resubscribes by itself.	If the publisher is legitimately quiet for longer, raise **Stale after (ms)**, or set it to 0 to turn the watchdog off. If diagnostics say *connected, not subscribed — retrying*, the broker is refusing the subscription; the reason follows in brackets.
The node at … has no activated configuration yet, so there is nothing to acquire.	The node exists but no configuration has been activated on it.	Configure connections and tags in the Desktop and save them to the node.
Everything works, but it is not your plant	A new node is commissioned with an example line (one simulated connection, ten signals) so that it shows something.	Remove it in **Acquisition ▸ I/O Configuration…** when you configure the real plant. The commissioning report refuses a node that still carries only the example line.
An alarm never raises	The rule is disabled, its condition does not compile, or the raise delay is longer than the condition holds.	Open **Acquisition ▸ Alarms…**: check **Enabled**, the *Condition is valid.* line and **Compiles to**. A rule with a bad condition is disabled at load, and the event log says *Alarm '…' disabled - bad condition: …*.
A client workstation never shows short alarms	Its credential at the node lacks the permission to read events, so the panel cannot sweep what the node recorded.	Re-save **Administration ▸ Deployment…** to mint a credential that has it.
If your installation includes the Web HMI module: an automation rule has never run	Its trigger names a signal this node does not acquire, or a service it does not have.	**Automation ▸ Repairs** names which; the editor also marks the step **Not on this node**.

The historian is unavailable

The historian's state is shown on the Desktop's status bar whenever a historian is configured, and stays there until it is no longer true.



The Desktop acquiring from a node while the status bar reads, in red, Historian: unavailable — This workstation has no credential for the node's data API, so it can show no history. Open Deployment and save the connection again to ask the node for one.

Status bar says	Means	What to do
(nothing about the historian)	No historian is configured. Live values work. If the node records continuously, trends show what the recordings hold; otherwise every trend starts when the screen opened.	Configure one in **Data ▸ Historian…** if you need history beyond the recordings' retention.
`Historian: <provider>`	Configured and storing.	Nothing.
`Historian: <provider> — spooling … sample(s), spool …% full`	The database is not taking data; samples are being held in the spool and will be delivered when it returns. Nothing is lost yet.	Restore the database. Loss begins only if the spool fills.
`Historian: <provider> — the spool cannot be written to`	The database is down and the spool's own disk is refusing writes. This is where loss starts.	Free space on, or repair, the spool's disk, and restore the database.
`Historian: unavailable — <reason>`	Configured and not running; samples offered to it are not being kept. The reason says why.	Act on the reason (see below).
`…; … refused sample(s) kept in <path>`	The database refused a batch outright. The samples and the database's reason are kept in a dead-letter file beside it.	Fix what the reason names, then replay the file.
`… (… source(s) unavailable)`	A historian source the store reads from is not reachable.	Check that source's connection in **Data ▸ Historian…**.

Common reasons after `Historian: unavailable —`:

Reason	What to do
This workstation has no credential for the node's data API, so it can show no history. Open Deployment and save the connection again to ask the node for one.	On a client workstation: **Administration ▸ Deployment…**, then save. The workstation mints its own credential at the node.
This node's configuration has no historian, so this host serves live values and no history.	Turn a historian on in **Data ▸ Historian…** (or the equivalent historian settings in the browser, if your installation includes the Web HMI module).
A provider that is not licensed or not installed	If your installation includes an optional historian module (SQL Server, Oracle, TimescaleDB, QuestDB or ClickHouse), check that it is licensed, enabled in **Administration ▸ Modules…**, and installed. Otherwise use the built-in provider.
The database cannot be reached or refuses the sign-in	Check the connection settings and credentials in **Data ▸ Historian…**, and that the database server is up. Schema creation needs rights the running service deliberately does not have; do it from **Data ▸ Historian…** as a signed-in engineer.
There is a historian at …, left by a version of this host that kept its stores one directory up.	An older layout's store is still on disk. Move or migrate it rather than leaving two.

Something is unavailable

A menu item that is disabled, a screen that refuses, or a feature that is missing altogether usually has one of four causes: the licence, the module switch, the files not being installed, or your role.

What you see	Why	What to do
Not permitted: *Your role (…) is not permitted to: …*	Your account's role does not include that action. Viewers may acquire, trend, play back and export; Operators may also acknowledge and shelve alarms; Engineers may configure; Administrators may also manage accounts, licensing, modules and backup.	Ask an administrator to do it, or to change your role.
Action blocked: *The action was blocked because it could not be recorded in the audit trail: …*	Every configuration change must be written to the audit trail; the trail could not be written, so the change was refused.	Fix what the message names (usually disk space or permissions on the state directory).
A module says *…is licensed and this site has not enabled it*	You own it; this site has it switched off.	**Administration ▸ Modules…**. Switching a module on activates a configuration revision; no restart is needed.
A module says *…this installation's licence does not include …*	The licence does not cover it.	Ask your supplier for a licence that includes it.
A module says *…is not installed on this machine*	Licensed and enabled, but its files are not here.	Install it: **Install…** in **Administration ▸ Modules…** in the Desktop (or the equivalent in the browser, if your installation includes the Web HMI module) with the signed `module.json` you were sent.

What you see	Why	What to do
A module is licensed but still unavailable	Several conditions gate a module, and the screen names the first that failed.	Usually the entitlement file was not installed, or the module is installed but not switched on.
A feature is missing entirely	The Community edition does not contain the vendor drivers and some modules; they are absent, not disabled.	Install a paid edition. A licence cannot add code that is not in the package.
License for … expired on ….	The licence has expired. Nothing is deleted; acquisition stops at the edition's limits.	Install a current licence in **Help ▸ License…**.
Commission its node first, or use a license that is not node-bound.	A node-locked licence needs the node it is locked to.	Commission the node, then quote the id the host prints (*Licensing is bound to this node: … id …*) when requesting a licence.
Tags or clients beyond a limit are refused on a trial	The trial carries every capability and module for 30 days, limited to 128 tags and 2 concurrent clients.	Install a full licence.
The data-access API answers HTTP 402	The licence lacks a capability. It is not a payment prompt and retrying will not help.	A person must change the licence.
The data-access API answers HTTP 429	This identity exceeded its per-minute request budget.	Back off and retry.
The data-access API answers HTTP 401 mid-session	The token was revoked; revocation takes effect on the next request.	Issue a new token.
Real-time endpoints answer `no_live_feed` (503)	Only a host that acquires has live data; a history-only server has none to serve.	Point at the acquiring host.
Recording ▸ Continuous records to one file: *The … edition records to a single file; continuous rotating capture is not included.*	Continuous capture is a Professional feature.	Use single-file recording, or upgrade the licence.
Acquisition ▸ Node history… says the installation keeps its configuration in a project file	Only a node keeps revisions.	Make the installation a node through **Administration ▸ Deployment…**.
File ▸ Export Timing Evidence (CSV)… is disabled	It works only on a recording.	Open a recording first (**File ▸ Open Recording…**).
If your installation includes the Studio Assistant module: the Assistant panel says it is unavailable	One of licence, switch or provider withheld it; the panel says which.	Fix the one it names.

Something is slow

What you see	Why	What to do
Live trends stutter in the Desktop	**View ▸ Scroll live charts continuously** repaints every live chart 25 times a second.	Turn it off, or turn on **View ▸ Draw live traces from kept columns**.

What you see	Why	What to do
A trend with many points is slow	A trend holding a million points is holding a million points.	Zoom to the window you need; use the historian's rollups for long ranges.
If your installation includes the Web HMI module: a page is slow in the browser	The page subscribes to too many detailed updates, or its trends scroll continuously.	Ask the page's author to use value tiles rather than detailed ones, and to turn continuous scroll off on busy charts.
A browser tab that was in the background comes back with a gap	Not a fault: missed frames are reported as a gap rather than hidden.	Nothing to fix.

Recovering

- **A recording outlives the session.** `.opslog` files describe their own signals and can be read while they are being written; a crash does not cost you the file. Open it with **File ▸ Open Recording…**.
- **A configuration change went wrong. Acquisition ▸ Node history…**, select the last good revision, **Roll back to this**. The rollback is itself a revision and can be undone.
- **A machine has to be replaced. Administration ▸ Back up / restore node…** takes the node's configuration revisions, accounts, licence, historian settings and audit log as one archive and restores it onto the replacement. Sample data is the historian's and is backed up with its database.
- **Back up the whole state directory**, including `secrets\master.key`. Secrets are sealed with a key bound to the machine, so a restore onto different hardware needs them (driver, mail, certificate and API passwords) re-entered. Rehearse the restore before you depend on it.
- **Before rotating the encryption key** (**Administration ▸ Rotate encryption key...**), take a backup.
- **A support bundle** (**Help ▸ Save Support Bundle...**) collects logs and settings with credentials masked, and deliberately excludes your projects, the user store, the audit directory and the key store. Read it before sending it.

Where logs live

Log	Where
Desktop application log	`<state>\logs\meelmo-YYYYMMDD.log`, one file a day, kept 14 days. **Help ▸ Open Logs Folder** opens the folder; **Help ▸ About Meelmo…** shows its path.
Event Log panel	In the Desktop window (**View ▸ Event Log**); what happened in this session.
Audit log	`<state>\audit\audit-YYYYMMDD.jsonl`, one hash-chained file per day. Kept every day by default; an administrator can set a retention of at least 30 days.
Service hosts (recorder service, node host, and any web or data-access host your installation includes)	Their console output, or the operating system's service log. Each states at start what it is attached to, what it serves and what it refused.
Historian dead-letter file	`dead-letter.log` beside the file of refused samples named on the status bar.

What to send with a report

1. The product version from **Help ▸ About Meelmo…**, and the edition and expiry from **Help ▸ License…**.
2. The deployment shape (standalone, server or client) from **Administration ▸ Deployment…**, and the node id the host prints.
3. A support bundle (**Help ▸ Save Support Bundle...**).
4. The host's console output from start, for anything about a service.
5. For anything about a screen: which screen, and what the status bar said at the time. For a chart: the project with a note at the moment in question.