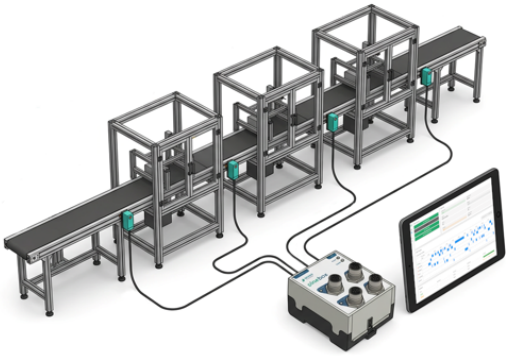
# 7. Line Configuration

User manual

www.pinetek-networks.com



CedarInsight

Pinetek Networks GmbH, Bachstelzenweg 8, 88677 Markdorf
info@pinetek-networks.com

2026-07-29

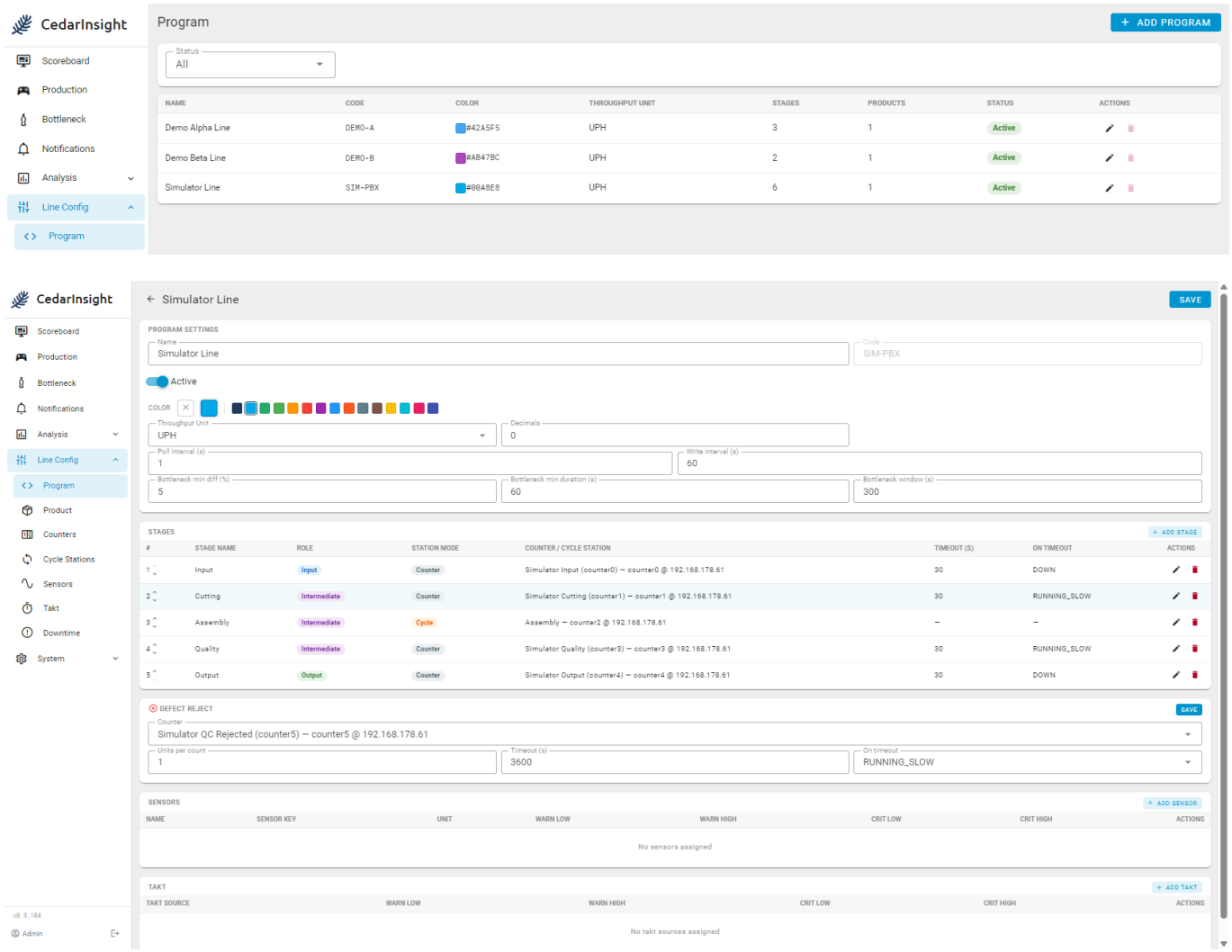
# Table of Contents

# 7. Line Configuration

*Line Config* describes your production line to CedarInsight. Required role: **Supervisor**.

Recommended configuration order: *Counters → Program (with stages) → Products*, then optionally Sensors, Takt, Cycle Stations and the Downtime catalogue. See also the commissioning checklist in chapter 3.6.

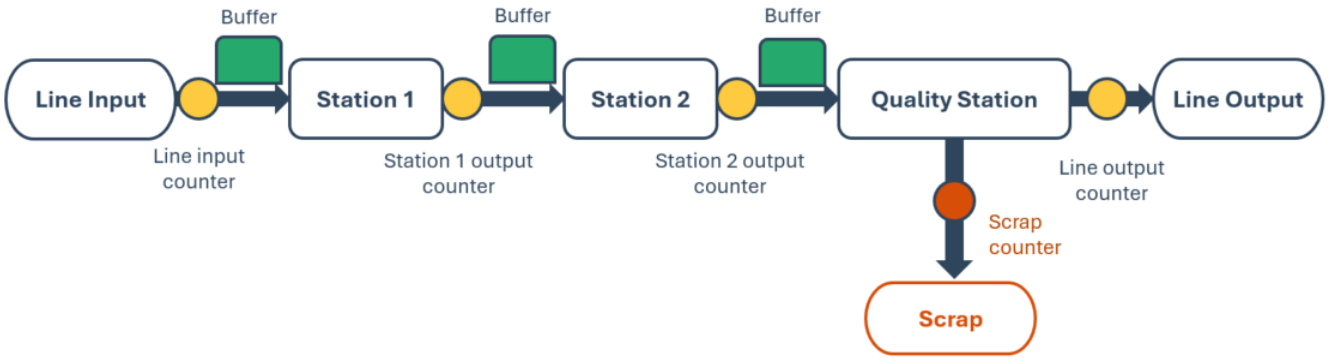
## 7.1 Programs







*Line Config → Program* defines line configurations. Click a program to open its detail view with the tabs **Stages**, **Sensors**, **Takt**, **Products** and **Program Settings**.




Programs usually have stages that are connected to counters or cycle stations:

- An Input stage (defined by the first position in the list)
- Several processing stages with counting between the stages
- An Output stage (defined by the last position in the list)
- Optional: one or more scrap counters

The minimal configuration is one output counter. The counter inputs are used to calculate the OEE values.

The program can also include takt and sensors. Those inputs do not have impact on the OEE calculation, but can be used for correlating in the production analysis or via the AI analysis (see below) .

### Program settings

| Field | Meaning |
|---|---|
| **Name / Code** | Display name and short code (code is shown on the Scoreboard). The code cannot be changed after the first production run |
| **Color** | Default color for this program's runs |
| **Cycle Time (s)** | Default ideal cycle time (can be refined per product) |
| **Throughput Unit** | Units per hour / minute / day; affects throughput displays; the actual throughput is defined at the product |
| **Decimals** | Decimal places for throughput values |
| **Poll interval (s)** | How often counter values are sampled (default 1) |
| **Write interval (s)** | How often sampled readings are written to the database (default 60) |
| **Bottleneck min diff (%)** | A stage must be at least this much slower than the fastest stage to count as deviating |
| **Bottleneck min duration (s)** | The deviation must persist this long to become a confirmed bottleneck |
| **Bottleneck window (s)** | Averaging window for throughput comparison |

### Stages

Each stage is one counting point of the line, in process order:

| Field | Meaning |
|---|---|
| **No.** | Position in the process (1 = first), automatically determined |
| **Stage name** | e.g. “Filler”, “Capper”, “Packer” |
| **Role** | *Input*, *Output*, *Defect / Reject* or *Intermediate* |
| **Counter** | The Pinebox counter or cycle station mapped to this stage (registered in 7.3) [PBX] |
| **Units per count** | Multiplier, e.g. 6 if one counter tick equals one pack of 6 units |
| **Timeout (s)** | No count for this long → stage considered stalled (drives Down detection, chapter 2.3.2) |
| **On timeout** | Behaviour when the timeout fires |

Roles matter for OEE: the **Input** stage counts *Units In* (actual quantity), the **Output** stage counts *Units Out* (good quantity), a **Defect/Reject** stage counts rejected parts, and **Intermediate** stages are used for throughput and bottleneck analysis.

A stage can alternatively run in **Cycle mode** instead of Counter mode; it is then fed by a cycle station (section 7.4) rather than a piece counter.

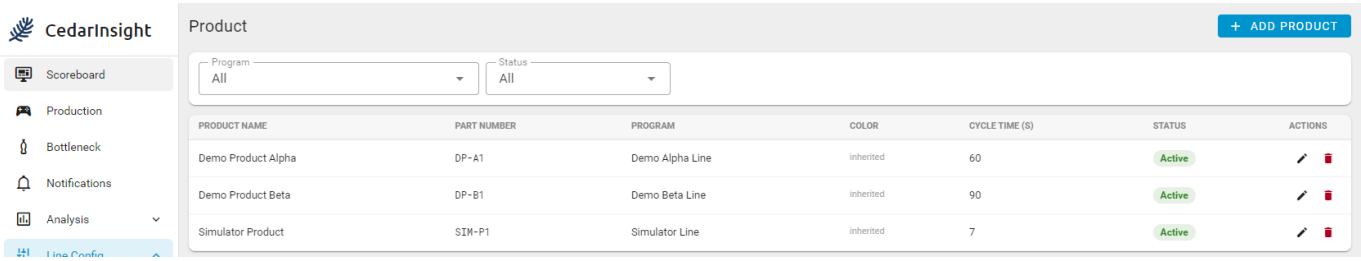
### Sensors / Takt tabs

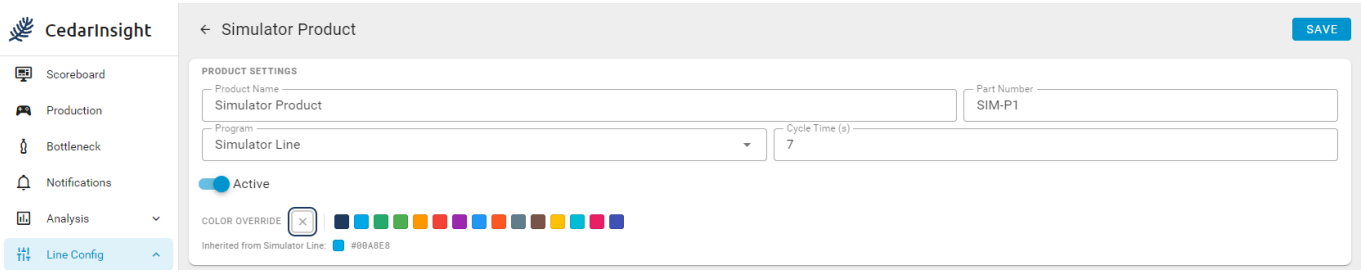
Assign registered sensors (7.5) and takt sources (7.6) to the program and set warning/critical thresholds (*Warn low/high*, *Crit low/high*). Assigned sensors and takt appear in the run trend chart (chapter 6.3). The recorded data is used to correlate takt times and sensor values with the throughput of the line, e.g. if air pressure drops, the takt times get longer.

The sensor data and takt time is recorded but does not contribute to the OEE calculation.

## 7.2 Products

*Line Config → Product* manages the articles you produce. A product runs on a defined program, but with it's own cycle time. Example: Filling lines that use the same stations for filling and labelling, but fill different container sizes (this different speeds and cycle times).



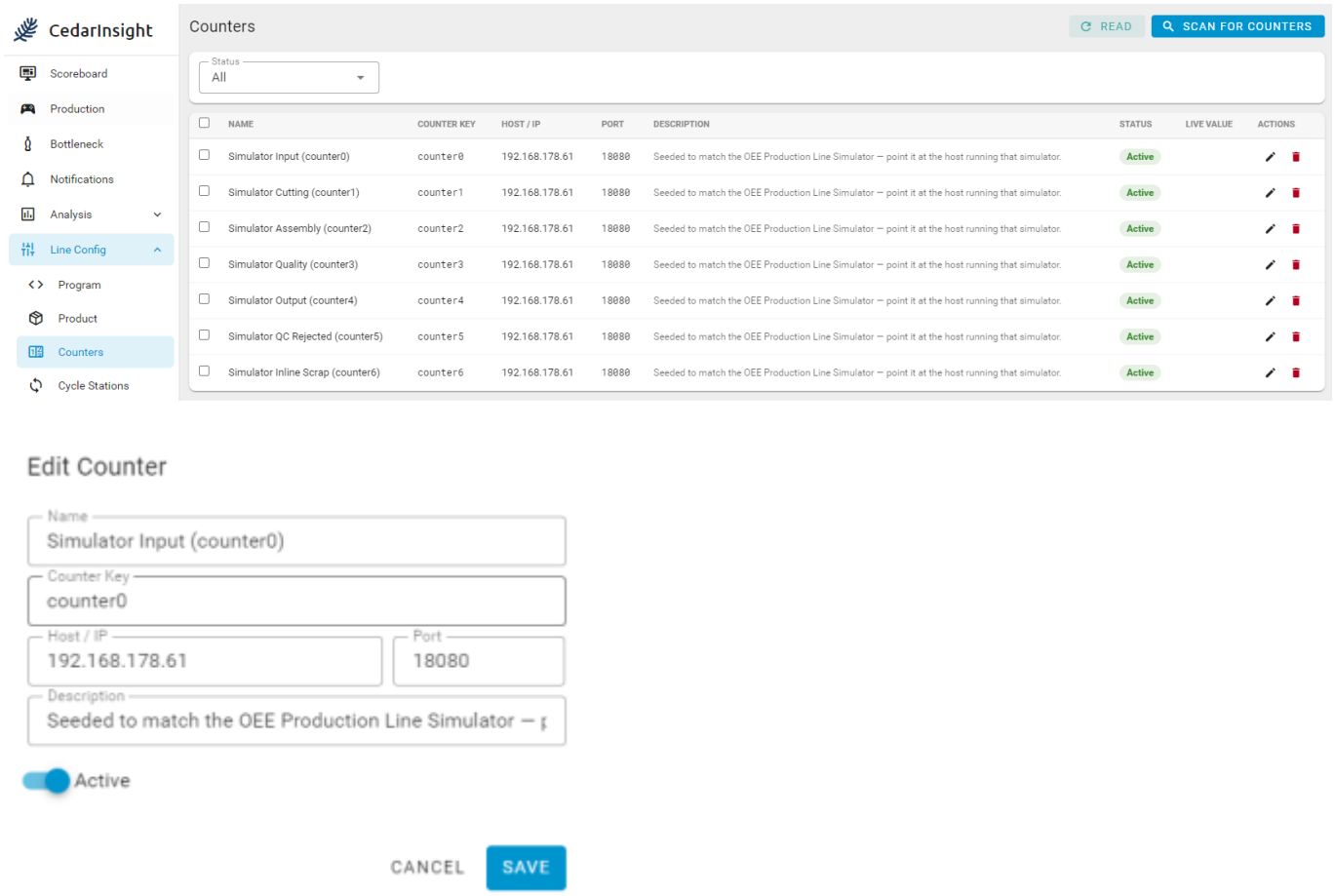

Per product:

| Field | Meaning |
|---|---|
| **Product Name** | Display name (mandatory) |
| **Part Number** | Your article/part number |
| **Program** | The program used to produce this product (mandatory) |
| **Cycle Time (s)** | *Ideal* cycle time per unit, the basis of the Performance calculation (chapter 10.3) |
| **Color Override** | Optional colour for timelines/charts; by default the product inherits the program colour |

An accurate ideal cycle time is essential; a value set too high makes Performance look better than it is, and vice versa.

## 7.3 Counters

*Line Config → Counters* is the registry of Pinebox piece counters.The counter values are used as input to the programs. In order to use counters, they need to be defined and configured in the Pinebox.



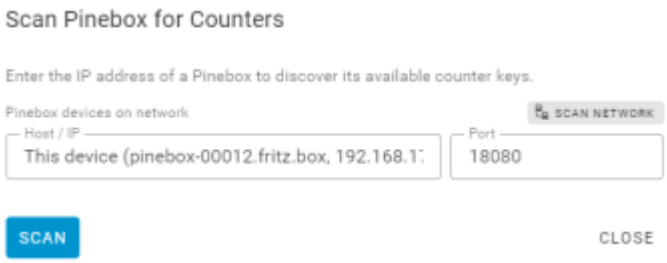




Each entry stores:

| Field | Meaning |
|---|---|
| **Name** | Display name |
| **Counter Key** | The key exactly as it appears in the Pinebox counter data [PBX] |
| **Host / IP, Port** | Address of the Pinebox providing this counter (default port `18080`) |

Rather than typing keys manually, use the scan functions:




1. **Scan for counters:** reads the available counter keys from a selected (or manually entered) host.
2. **Scan network:** discovers Pinebox devices on the local network for remote counters
3. **Import** the keys you need; already registered keys are marked.

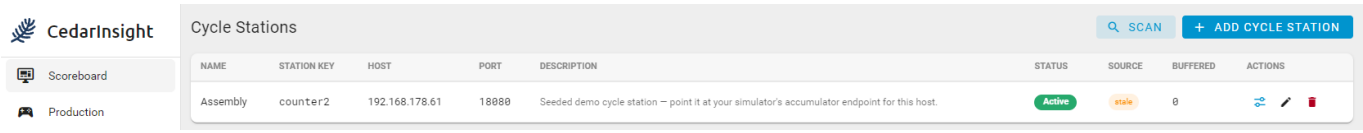
**Read** fetches the live values of all registered counters; use this to verify the mapping (unreachable hosts are reported).

Which physical input feeds which counter key is configured on the Pinebox, see the Pinebox Manual [PBX].
The default port on the Pinebox is 18080

## 7.4 Cycle Stations [PBX]

Cycle stations monitor machines whose cycles are detected from their power consumption (e.g. presses, moulding machines): the Pinebox measures electrical power/energy, and a detector recognises each machine cycle from the power rise at cycle start and the power fall at cycle end.

In the list of cycle stations, the detected buffers are shown and can be updated




### 7.4.1 Registering cycle stations

*Line Config → Cycle Stations* manages the registry (pen symbol). Cycle stations serve as input to the program counters. To use cycle stations, they need to be configured in the Pinebox.


Per station:

| Parameter | Meaning |
|---|---|
| **Name** | Station name for identification |
| **Station Key** | Identifier on the source device (Pinebox) |

| Parameter | Meaning |
|---|---|
| **Host/Port** | Pinebox IP or hostname and port |
| **Description** | Internal description of the cycle station |
| **On stall** | *see below* |

As with counters, **Scan** discovers devices on the network and imports the stations they advertise.

**On stall** defines what happens when the detector flags a stalled cycle:

- **Auto-scrap:** the part is automatically counted as scrap,
- **Ask operator:** a resolution prompt appears; the operator chooses *Scrap*, *Complete* or *Re-enter*,
- **Auto-continue:** the partial cycle is accepted as complete.

### 7.4.2 Calibration


The **Calibrate** view shows a live power scope (~2 min window; scroll to zoom, drag to pan) with the average power and its rate of change, plus the currently configured thresholds and the observed idle/peak values.

Detector parameters:

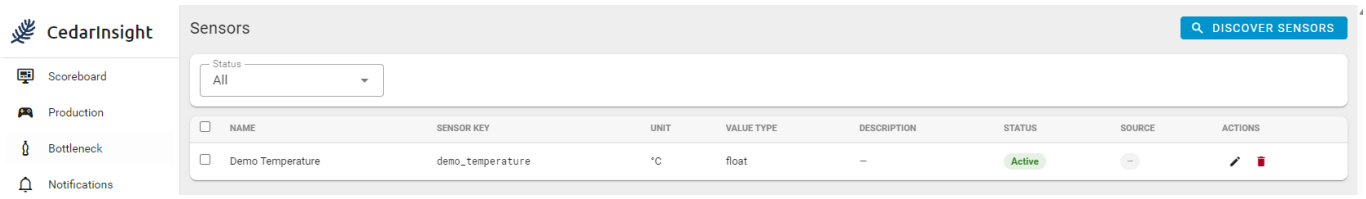
| Parameter | Meaning |
|---|---|
| **Baseline (W)** | Machine idle power draw, used for stall detection |
| **Start rise (W/s)** | Minimum rate of power rise to begin a cycle (right scale) |
| **End fall (W/s)** | Minimum rate of power fall to end a cycle (right scale) |
| **Min energy (kWh)** | Minimum accumulated energy before the end condition may fire (noise guard) |
| **Stall timeout (s)** | Average power below baseline for this long → cycle flagged as stalled |
| **Slow timeout (s) / Max cycle (s)** | Limits for slow/overlong cycles |

| Parameter | Meaning |
|---|---|
| **T min / T max (s)** | Optional duration gate , cycles outside these bounds are not accepted as OK |

Calibration procedure:

1. Let the machine run a few normal cycles.
2. Watch the **rate trace** (right axis, W/s): set **Start rise** to the leading-edge peak and **End fall** to the trailing-edge dip of a cycle.
3. Set **Baseline** to the observed idle power.
4. Use the energy measurement to determine **Min energy**: click once on the curve to set the start marker, click again for the end marker; the energy under the curve and the time span are displayed. Set **Min energy** safely below a normal cycle's energy but above noise level.
5. Save and verify that the **Detected Cycles** markers match the real machine cycles.

## 7.5 Sensors [PBX]




### How sensor data reaches CedarInsight

Sensors are the one data type in CedarInsight that is pushed, not polled; everything else (counters, cycle stations, takt) is fetched by CedarInsight itself on a timer, and sensors instead arrive whenever the source decides to send them. The path is:

1. **Pinebox**: the IO-Link sensor data is configured and poweredread at the hardware/fieldbus level. [PBX]
2. **Node-RED**: a flow running on the Pinebox (local or remote) cyclically (via cyclic inject) reads that value as described in the Pinebox manual, packages it into the JSON shape CedarInsight expects, and sends it on.
3. **CedarInsight**: serves the receiving endpoint and stores what arrives. Nothing needs to be configured on the CedarInsight side to make it “listen”; the API is always up as part of the normal backend, and the only setup is registering the sensor here in **Line Config → Sensors** so incoming keys are recognised.

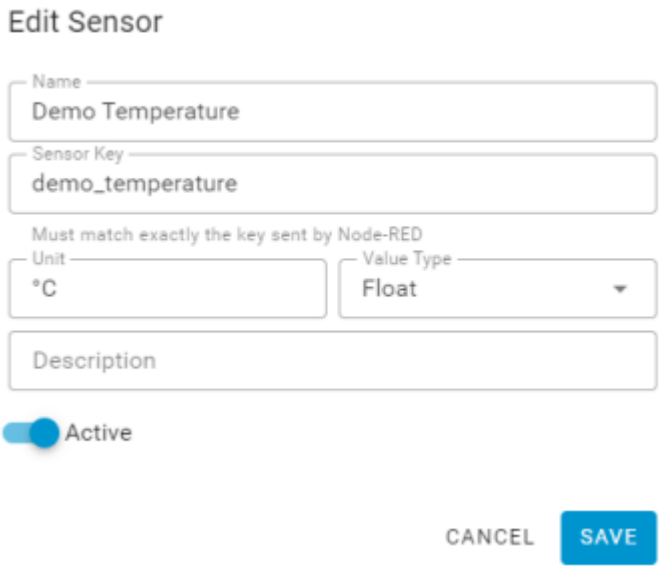
```
POST http://<cedarinsight-host>:8000/api/sensors/ingest
```

sent as a single sample object or an array of them, e.g.

```
''[{"sensor_key": "line1_temp", "value": 74.2, "unit": "°C"}]''
```

The full payload contract, field-by-field, and a worked Node-RED flow example (input node → function node → HTTP request node) are in chapter 11.6 [PBX]. This section covers the registry side: what to configure here once data is flowing.

### Registering a sensor




Per sensor: **Name**, **Sensor Key** (must match the `sensor_key` the source sends, exactly; this is the only link between the incoming push and the registered row), **Unit**, **Value Type**, description.

Use **Discover sensors** to list keys currently sending live data (i.e. anything that has POSTed to the ingest endpoint recently, whether registered yet or not) and **Register** them directly, including their current value, so you can identify each signal before naming it.

If a not-yet-registered key's push includes `display_name`/`unit`, the sensor is auto-registered on first arrival; Discover will simply show it as already present. Once a sensor is registered and named through the UI, pushed `display_name`/`unit` values never overwrite what's set here, the UI always wins.

### Checking that data is actually arriving

The **Source** column in the sensor list reflects whether a sample for that key has arrived recently (within the last 30 seconds):

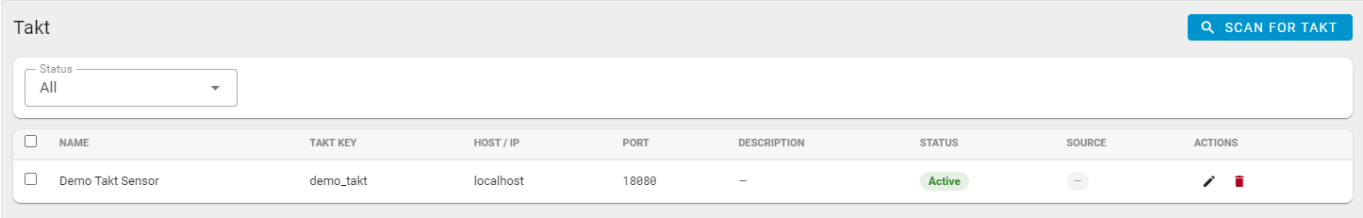
- **OK** (green): a sample was received within the last 30 seconds.
- **stale** (amber): no sample for longer than that; hover the chip for the last-seen time. Registered sensors keep their configuration regardless; this only reflects whether the push side is currently alive.
- **—** (grey): no sample has ever been received for this key since the backend last started.

This is independent of whether a production run is active; Node-RED can push (and this column will show **OK**) even with the line stopped.

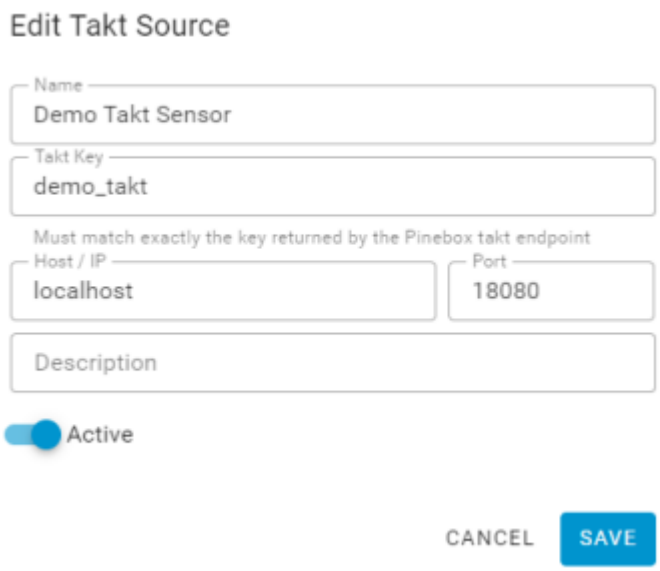
## 7.6 Takt

Takt is the cycle time of one individual unit (e.g. robot) how long that one process took, timed directly by the Pinebox, not derived from a counter. Takt is the time measurement between two events, e.g. a movement from A-to-B with a sensor on both ends.

Takt is complementary information, not a driver of OEE: unlike Counters and Cycle Stations, takt values never feed into Availability, Performance, or state transitions; like sensors they exist purely to catch a single slow (or suspiciously fast) cycle in the moment, via the **warn/crit thresholds** set on the program assignment (7.1), which raise notifications when breached and get overlaid against actual values in the run trend chart (6.3) for root-causing dips seen in the “real” OEE numbers.



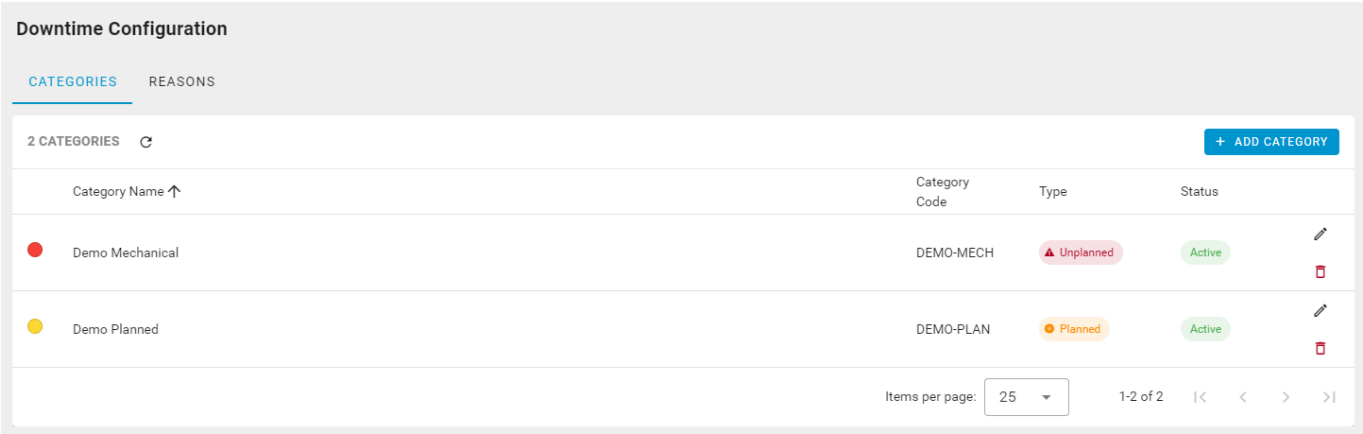

*Line Config* → *Takt* registers takt (cycle-time) sources measured by Pinebox timers. Per source: **Name**, **Takt Key** (must match the key of the Pinebox takt endpoint exactly [PBX]), **Host/Port**, description.



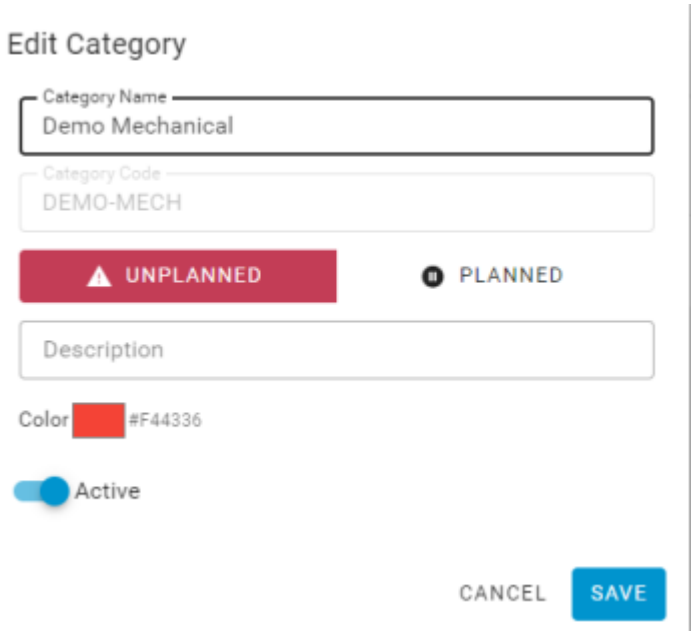

**Scan for takt** reads the available timer keys from a Pinebox (local or remote) and registers them directly.

Assign takt sources to a program (7.1) to monitor cycle times against thresholds and overlay them in the run trend chart.

## 7.7 Downtime reason catalogue




*Line Config* → *Downtime* maintains the two-level reason catalogue used when classifying stops (chapter 5.4): **Categories** (e.g. “Mechanical”, “Material”, “Organisational”) each containing **Reasons**.




| Field | Meaning |
|---|---|
| **Category Name** | Display name of category/reason |
| **Category Code** | Unique short code, e.g. `MECH-FAIL`, cannot be changed after creation |
| **Type** | Planned / unplanned classification |
| **Color** | Color used in charts |
| **Sort Order** | Display order in the selection dialog |

Keep the catalogue short and unambiguous; operators must find the right reason in seconds. Prefer ~5–10 reasons per category over an exhaustive list.