The Historian Trigger Problem Nobody Talks About

Why your end-of-shift snapshot is quietly lying to you, and what triggering correctly looks like

A historian fires a trigger at the end of a shift and writes a row to the database. That row is wrong more often than anyone wants to admit.

Triggering is the single most important thing a historian does, and it is the least understood. Most engineers picture a trigger as a simple stopwatch: the clock hits a value, the historian saves data, done. That mental model is what gets you into trouble. A trigger is not just a signal to write a row. It is a decision about which data is captured and whether that data is consistent at the instant of capture. Get the timing right and you have a clean, time-stamped snapshot you can trust for reporting, analytics and AI. Get it wrong and you have a row of numbers that never actually existed together on the plant floor.

This brief is the result of issues that real users of both the RTA Historian product and other historians have reported. It describes how historians handle triggering, the trigger supported by the RTA Historian and a design flaw buried in most historians that corrupts data at the exact moment you care about most.

What Does a Trigger Do in a Historian?

A trigger tells the historian when to act on a data model, either by storing it in the database or publishing it to an external system. It is the timing engine that governs both capture and publishing.

The RTA Historian continuously reads the values defined in the data models a user loads. A data model is a collection of related values, and those values can come from very different places at the same time. A single model might hold a temperature from a chiller on a BACnet network, a weight from a scale on a Modbus network and a dozen tag values pulled from Allen-Bradley PLCs. The historian is always scanning those sources in the background.

The trigger is what turns that continuous scan into a discrete event. Two things can happen when a trigger fires:

- **Capture** — the historian writes the current data model into the time-series database as a stored record.
- **Publish** — the historian pushes the data model out to an external destination: SQL to a database, a JSON payload over MQTT, OPC UA, a REST call over HTTP, a WebSocket, email, FTP or a CSV or PDF written to a USB stick.

Same trigger engine, two jobs. That separation matters, because it means the same event that archives a record locally can simultaneously feed a Unified Namespace or a cloud analytics stack without a second mechanism bolted on.

What Trigger Types Does the RTA Historian Support?

The Historian supports four trigger types: cyclic interval, change-of-value, schedule and USB insertion. Each map to a different real-world reason you would want to capture or move data.

No single trigger type covers every application, so the Historian offers all four. The table below lays them out.

Trigger Type	Fires On	Typical Use
Cyclic / Milliseconds Interval	A fixed time interval, from 1 ms up to roughly 65,000 ms	Archiving a continuous profile, such as oven temperatures across a curing process
Change of Value (CoV)	A change in a Float, Integer, Boolean or String value	End of job, part present, alarming and diagnostics — capture only when something moves
Schedule	A calendar date, day or time, down to the minute	Routine captures: every Tuesday at noon, first Wednesday at 6 pm, every day at 6 am
USB Inserted	A USB drive being physically inserted into the Historian	Grab-and-go pulls to CSV or PDF, or firing an email when someone plugs in

Table 1 — The four RTA Historian trigger types.

Cyclic Triggers – These are the simplest triggers. Few people fail to understand how a cyclic trigger works.

Change-of-Value trigger – There are always questions on this when float values are used. A float will trigger any change and that is rarely what you want on a noisy analog signal. So, CoV triggering on a float carries a hysteresis band: you define how far the value has to move before it counts as a change. Integers, Booleans and strings trigger any change from the previous value. The CoV trigger also carries a startup inhibit time, a delay in seconds that keeps the Historian from firing a storm of triggers while values are still settling after a reboot.

Schedule Trigger – Schedule triggers are set using five parameters: the Month, the Day of the Month, the Day of the Week, the Hour and the Minute. These can be combined to form many different schedules such as 2nd Tuesday at noon, 6pm every day and the first of January and July at midnight.

Set up the trigger by selecting:

- The Trigger Month – Select every month, a single month or multiple months (April and October).
- The Trigger Day of the Month – Select every day, multiple days, a single day or a series of days in the month (i.e. the 3rd through the 6th).
- The Trigger Day of the Week – Select every day of the week, a single day of the week or multiple days in the week (i.e. Tuesday & Thursday).
- The Trigger Hour of the Day – Select the hour or multiple hours to trigger.
- The Trigger Minute of the Hour – Select the minute or multiple minutes (i.e. every quarter hour with 0, 15, 30 and 45).

You could set a trigger to fire only on a Wednesday, when that Wednesday falls on the 7th of August.

Example Schedule triggers for the RTA Historian are illustrated in the following figures:

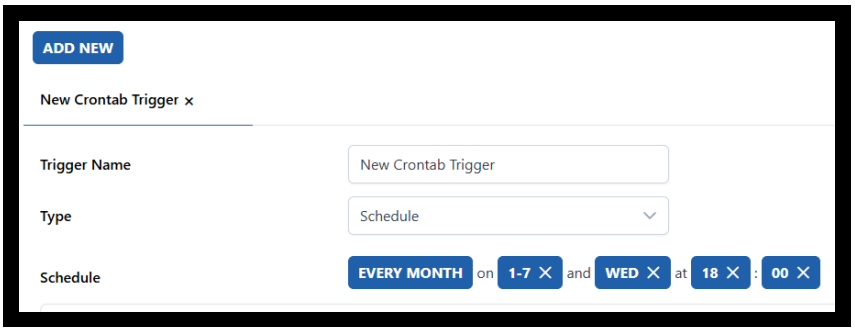

Figure 1 - Schedule a Trigger on the 1st Wednesday of every month at 6pm. The 1-7 indicates that the Wednesday can occur on the 1st through the 7th.

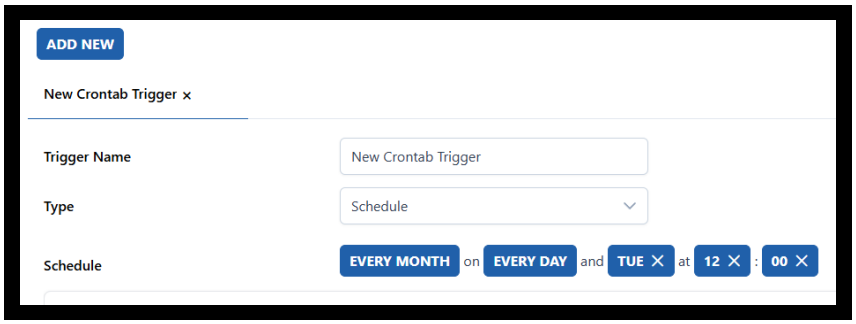

Figure 2 -Schedule a Trigger Every Tuesday at Noon. The day of the month doesn't matter. Trigger day is every day.

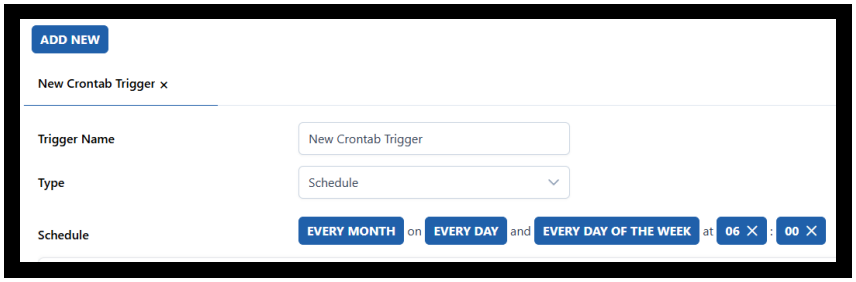

Figure 3 - RTA Historian, Schedule Trigger every day at 6AM

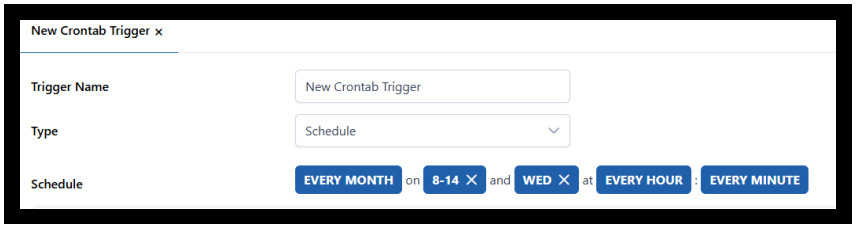

Figure 4 - Schedule a trigger on the 2nd Wednesday of the month. 2nd Wednesday is the 8th through the 14th.

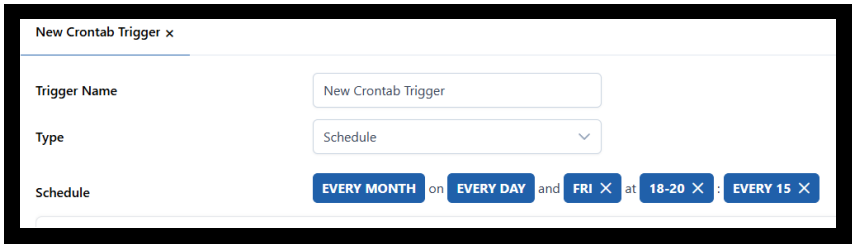

Figure 5 - Schedule a trigger for Friday night every 15 minutes between 6 and 10pm.

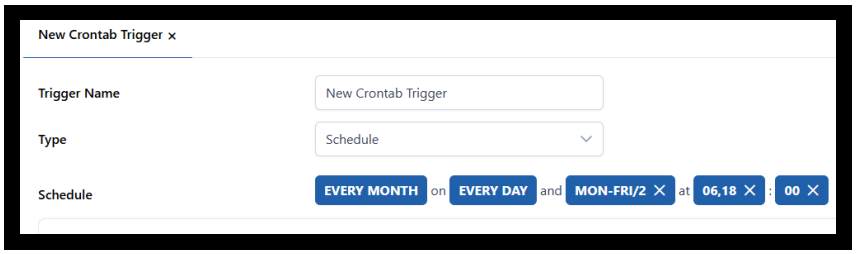

Figure 6 - Schedule a trigger for Monday, Wednesday, Friday at 6am and 6pm.

Can a PLC Tag Gate a Trigger?

Yes. All triggers can be enabled or disabled by a conditional value. When the conditional value is true, the trigger is enabled. When false, the trigger is disabled. A conditional is an expression that evaluates to a true/false Boolean value (i.e. Tag1 = Tag2, Tag1 > 50, TagBoolean1, etc.)

This is more useful than it first sounds. A schedule trigger set for every day at 6 am is fine until the line is down for maintenance and you are archiving a row of meaningless idle values. Attach a conditional value — fire only when the PLC tag HST_Counter is true — and the trigger respects the actual state of the process. The clock proposes and the PLC disposes. The Historian is not blindly writing rows on a timer. It is writing rows when the plant floor agrees they are worth writing.

The conditional value gates both data storage and publishing. A tag value can be used to enable/disable publishing to an external database. For example, you could publish to the external database only when the machine is rejecting product.

What is the Hidden Flaw in Most Historian Triggering?

Most historians save whatever values they happen to be holding at the instant a trigger fires. Some of those values are fresh and some are stale, so the saved record is a mix of data that never coexisted in reality.

Here is the mechanism. A historian scans its tag set on a rolling basis. It reads tag 1, then tag 2, then tag 3 and so on, cycling through the set continuously. When a trigger fires mid-cycle, a naive historian simply grabs the current contents of memory and writes them. But those contents are patchwork. Some tags were updated microseconds ago. Others have not been refreshed since the last time the scan swept past them, which could be many milliseconds — an eternity on a fast line.

The result is an internally inconsistent record. Consider a trigger that fires at the end of a shift while the machine is still running:

Tag	Value Saved	Reality at Trigger
Motor Speed (RPM)	1,750	Read fresh — correct
Power Draw (kW)	41.2	Stale — last read 30 ms ago, actual value now 52.8
Vibration (mm/s)	2.1	Stale — spiked to 6.4 after last scan
Part Count	8,411	Read fresh — correct

Table 2 — One trigger, four tags, two of them wrong. The saved row describes a machine state that never existed.

Anyone reading that record later sees a motor at 1,750 RPM drawing only 41.2 kW with low vibration. A healthy snapshot. The truth was a power spike and a vibration event that the record erased. The row is not corrupt in any way a database integrity check would catch. It is worse than corrupt. It is plausible.

Why is Mid-scan Triggering so Dangerous for High-Frequency Data?

On fast-changing variables like power quality, vibration and machine speed, a few milliseconds of skew changes the number completely. Mid-scan triggering timestamps everything as one instant when the values actually came from several.

Slow signals forgive sloppy timing. A tank temperature that moves half a degree a minute does not care whether it was read 40 ms early. The variables that matter most for diagnostics, energy analysis and predictive maintenance are exactly the ones that do not forgive it. Power, vibration and speed can swing hard inside a single scan cycle. Those are the signals engineers reach for when they are hunting a fault or feeding a model, and they are the ones mid-scan triggering mangles worst.

Then it compounds. Feed these inconsistent snapshots into an automated report and the report is wrong. Feed them into an analytics dashboard and the trend line is built on states that never happened. Feed them into an AI or machine-learning model and the model learns from fiction. The failure is silent all the way down the stack,

which is what makes it dangerous. Nothing throws an error. The numbers just quietly stop matching the plant floor.

How does the RTA Historian Solve Mid-Scan Triggering Issues?

The RTA Historian rescans the entire tag set the moment a trigger fires, before it writes or publishes anything. Every value in the record is refreshed at the same instant, so the snapshot is internally consistent by design.

The fix is not exotic. The accepted way to solve this is to re-read the whole tag set on the trigger rather than trusting whatever is sitting in memory. The problem is that in many historians this behavior is an optional setting an engineer has to know to turn on, and in some it is not available at all. If you do not know the flaw exists, you never go looking for the switch.

The RTA Historian makes synchronized rescan standard behavior. When a trigger fires, the Historian re-reads every tag in the model and then commits the record. There is no mix of fresh and stale values because everything is fresh. The timestamp on the record describes a machine state that genuinely existed at that instant, not an average of several instants smeared together.

Put the two behaviors side by side:

- **Standard historian** — trigger fires, dump current memory contents, write the row. Fast, and quietly wrong on high-frequency tags.
- **RTA Historian** — trigger fires, rescan the full tag set, then write the row. Every value carries the same timestamp and the same truth.

For a control engineer watching power, vibration and speed, that is the difference between an accurately timestamped snapshot and a data gap that hides the exact event you built the historian to catch.

When are the Limitations of the RTA Historian?

The RTA Historian is a machine-level historian built for local, plant floor data capture on closed OT networks. It is not an enterprise-wide historian for consolidating every site into a corporate data lake.

Correct triggering is worth nothing if the tool is pointed at the wrong problem. Too many plants buy software the way people buy oversized pickup trucks — impressive, expensive and wrong for the actual job. The RTA Historian is a scalpel, not a Swiss Army chainsaw. Its job is to sit close to the PLC, capture consistent data at the machine and hand that clean data off to whatever lives downstream. If the requirement is a single enterprise historian spanning dozens of sites and corporate IT dashboards, this is the wrong tool and no amount of clever triggering changes that.

But that closeness to the PLC is the whole point. The most important context in an industrial system lives nearest the machine, and it starts to decay the moment data leaves the cell. A machine-level historian that captures consistent, correctly timestamped data at the source is doing the one job everything downstream depends on. Capture clean, or everything after it is built on sand.

Why Machine-Level Precision Matters

Triggering looks like a solved problem. It is not. The trigger type — cyclic, change-of-value, schedule or USB — determines **when** you capture and getting that right is the easy half. The hard half is **what** you capture at that instant, and most historians get it wrong by saving a mix of fresh and stale values that never existed together.

The RTA Historian closes that gap by rescanning the full tag set on every trigger as standard behavior, so the record you store or publish is a true synchronized snapshot. Match that to the right trigger for the job, gate it with a PLC conditional value where it helps and keep the tool where it belongs, at the machine. Do that, and the data

feeding your reports, your dashboards and your AI describes the plant floor as it actually was. Not as it plausibly might have been.

More information

Talk to an Enginerd about the RTA PLC Historian, email support@rtautomation.com, call 1-800-249-1612 or visit rtautomation.com/historian.