

Hub

The live order board: assigning, incidents, critical actions and alarms.

- [The Hub: what it is and who can do what](#)
- [The order board](#)
- [Order details and pickup code](#)
- [Assigning and unassigning a driver](#)
- [Changing the promised time](#)
- [Logging an incident](#)
- [Cancelling an order](#)
- [Forcing a delivery](#)
- [Stores and driver alerts](#)
- [Branch alarms](#)
- [Alarm settings](#)
- [Archived alarms](#)

The Hub: what it is and who can do what

“ **Audience:** hub operators, shift managers and operations managers working the live order board. **You need:** *Hub — orders and status* at minimum. Individual actions each need their own permission, listed below.

The Hub is the live operational view of orders in flight. Where the dispatcher decides automatically, the Hub is where a person intervenes: assigning a driver by hand, changing a promised time, logging an incident, or — rarely — forcing an order closed.

What the Hub is for

The dispatcher already assigns most orders on its own, every cycle. You open the Hub when something needs a human: an order the algorithm couldn't place, a customer calling about an ETA, a driver who never picked up, a store whose alarms are firing.

If you want to understand *why* an order was assigned the way it was, that is the dispatcher's logic rather than the Hub's — see [What the dispatcher does](#).

What each action needs

Access is granular on purpose. Holding the Hub view does not imply holding the critical actions.

Action	Permission	Who typically has it
See orders and status	<i>Hub — orders and status</i>	<code>hub_operator</code> and above
See an order's pickup code	<i>Hub — pickup code</i>	Kept separate from assigning, deliberately
Assign a driver	<i>Hub — assign</i>	<code>hub_operator</code> and above
Unassign a driver	<i>Hub — unassign</i>	<code>hub_manager</code>
Change the promised time	<i>Hub — modify ETA</i>	<code>hub_manager</code> , <code>branch_manager</code>

Action	Permission	Who typically has it
Log an incident	<i>Hub — incidents</i>	hub_manager, branch_manager
Cancel an order	<i>Hub — cancel</i>	hub_manager
Force delivery	<i>Hub — force delivery</i>	hub_manager
View driver alerts	<i>Hub — driver alerts</i>	hub_operator and above
View branch alarms	<i>Hub — branch alarms</i>	hub_operator and above
Archive alarms	<i>Hub — archive alarms</i>	hub_manager
Change alarm thresholds	<i>Alarm settings — edit</i>	alarms_config_manager

A `hub_operator` deliberately has the everyday actions but **not** force delivery, cancel or ETA changes. See [Access by section](#) for the full grid.

“ The pickup code is a separate permission from assigning, even though in practice the same people usually hold both. Keeping them apart means reading a pickup code never has to be bundled with the ability to move orders between drivers.

The pages in this chapter

Task	Page
Find an order, understand the board	The order board
Look at one order in detail	Order details and pickup code
Put an order on a driver, or take it off	Assigning and unassigning a driver
Change what the customer was promised	Changing the promised time
Record what went wrong	Logging an incident
Stop an order entirely	Cancelling an order
Close an order the system couldn't close	Forcing a delivery
See stores and driver alerts	Stores and driver alerts
See and act on store alarms	Branch alarms
Tune when alarms fire	Alarm settings

Task	Page
Review alarms already dealt with	Archived alarms

What you can and can't see

Everything in the Hub is filtered by your scope. A `branch_manager` sees their own store's orders; a chain-wide role sees all of them. Two colleagues looking at the same board can legitimately see different orders — see [How scope works in practice](#).

The order board

“ **You need:** *Hub — orders and status.*

The order board is the working list of orders currently in play across the stores in your scope.

Filtering by store

The main filter is **stores**: pick one, several, or leave it open for everything within your scope.

This is the filter to reach for first. A chain-wide role looking at every store at lunchtime is reading hundreds of rows; narrowing to the two stores actually having trouble makes the board usable.

“ The store filter narrows *within* your scope — it never widens it. If a store isn't in the list at all, that's scope, not filtering. See [How scope works in practice](#).

Identifying an order

Every order is identified by four things together, and you need all four when referring to one:

Part	What it is
Store	Which store the order belongs to
Ticket number	The order number at that store
Source	Where the order came from — web, mobile app, an aggregator, and so on
Brand	Which brand, when the chain runs more than one

A ticket number on its own is **not** unique across stores. This is why [Reporting a dispatcher issue](#) and any support conversation asks for store plus ticket number plus source rather than just a number.

Reading the board

For each order the board shows who it is currently with, what state it is in, and the promised time it is being measured against.

Two terms matter most, and they come from the dispatcher rather than the Hub:

- **Pending** — not yet assigned, but not yet late either.
- **Critical** — not yet assigned and already past its promised time.

Both are defined in the [Dispatcher glossary](#). An order sitting in Pending is normal between cycles; an order sitting in Critical is the board telling you to intervene.

When an order looks stuck

Before acting by hand, it is worth knowing whether the dispatcher has genuinely given up or simply hasn't run yet:

1. The dispatcher re-evaluates each store every cycle — by default no more than 60 seconds apart, and up to 300.
2. An order that failed to place in one cycle is retried from scratch in the next, including a fresh attempt with external carriers.
3. Only after that does manual intervention add anything.

The common reasons an order legitimately stays unassigned are listed in the [Dispatcher FAQ](#) — genuine capacity shortage, a zone that allows no available driver type, or incomplete address data.

“ **Manual assignment has a lasting consequence.** An order assigned by a person is permanently locked out of automatic reassignment: the dispatcher will never move it again, even if a much better driver frees up seconds later. Assign by hand when you mean to override the algorithm, not to hurry it along.

From here

- One order in depth → [Order details and pickup code](#)
- Put it on a driver → [Assigning and unassigning a driver](#)

- It's not going to make the promised time → [Changing the promised time](#)

Order details and pickup code

“ **You need:** *Hub — orders and status* for the detail view; *Hub — pickup code* to see the pickup code, which is a separate permission.

The detail view is the single screen worth screenshotting when you report a problem — it carries the timestamps and the current assignment together.

Opening an order

An order is identified by store, ticket number and source together — see [The order board](#). The detail view needs all of them, which is why you open it from the board rather than by typing a number.

What it tells you

You're asking	The detail view answers
Who has it right now?	The current driver or carrier, and since when
Is it going to be late?	The promised time against the current estimate
What has already happened?	Timestamps for each step, including arrival at the store where recorded
Has anything been logged against it?	Any incidents recorded on the order

The pickup code

The pickup code is the code a driver or carrier uses to collect the order.

It sits behind its own permission, separate from assigning. In practice the same people usually hold both, but keeping them apart means someone can be given the ability to read a code — to a driver on the phone, say — without also being able to move orders between drivers.

“ Treat the pickup code like any other collection credential. Read it to the person collecting; don't paste it into a group chat or a ticket, and blank it out of screenshots before attaching them to a support request.

Using it in a support request

If you're reporting something, this screen is the evidence that helps. Include:

- The **store, ticket number and source** — a ticket number alone isn't unique.
- The visible **timestamps** and **current assignment**.
- If the dispatcher is involved, the **Request ID** from the Scheduler, which points to one exact run.

What doesn't help is a screenshot of the board, or of an unrelated dashboard. Full checklist in [Reporting a dispatcher issue](#).

From here

- Change the driver → [Assigning and unassigning a driver](#)
- Change the promised time → [Changing the promised time](#)
- Record what went wrong → [Logging an incident](#)

Assigning and unassigning a driver

“ **You need:** *Hub — assign* to assign and to see available drivers. Unassigning is a **separate** permission, *Hub — unassign*, typically held only by `hub_manager` .

Seeing who is available

Open the order and ask for available drivers. The list is built for that specific order — it accounts for the order's store and its zone rules, so it is not simply "everyone on shift".

If the list is empty or shorter than you expect, the reasons are the same ones the dispatcher works from: capacity, zone restrictions on driver types, distance, carrier schedules. [When does an alternative exist?](#) lists the carrier checks in order.

Assigning your own driver versus a carrier

These are genuinely different operations, and the difference trips people up:

	Own driver	Third-party carrier
What you pick	The specific driver	The carrier company , not a courier
Who chooses the courier	You	The carrier does
What you can predict	Exactly who is going	Only that the carrier accepted it

With a carrier you are handing the order to a company. Which of their couriers turns up, and when, is theirs to decide — which is why carrier timing is governed by the limits in [External carriers](#) rather than by your choice.

Your own driver types are fixed. The carrier types available depend on what is configured and active for your brand, so the list differs between brands and can change without the Hub changing.

Assigning

1. Open the order — see [Order details and pickup code](#).
2. Request available drivers.
3. Pick an own driver, or a carrier.
4. Confirm.

“ **This is permanent in one specific sense.** An order assigned by a person is locked out of automatic reassignment for good. The dispatcher will not move it later, even if an idle driver appears at the store moments afterwards. That is the intended behaviour — a manual assignment is treated as a deliberate override, not a hint. See [When an order changes driver](#).

Unassigning

Unassigning takes the driver off the order and puts it back in play.

It is a separate, higher permission than assigning because it undoes someone else's action, including the dispatcher's. Use it when an assignment was made in error, or when the assigned driver has become unavailable and you want the order back in the pool.

After unassigning, the order is treated as unassigned again and the next dispatcher cycle will consider it normally.

“ Unassigning an order that is already **with a carrier** is not the same as cancelling with that carrier. If your intent is to stop the carrier, see [Cancelling an order](#), which handles the carrier side atomically.

What you cannot do

- **An order already out for delivery cannot be reassigned.** Once the driver has left the store, the dispatcher freezes it and so does the Hub. See [Known system boundaries](#).
- **You cannot assign outside your scope.** Orders and drivers are both filtered to what you can see.

Changing the promised time

“ **You need:** *Hub — modify ETA*. Held by `hub_manager` and `branch_manager` .

The promised time is what the customer was told, and what the system measures lateness against. Changing it is the right move when the original promise is no longer achievable and the customer has been told — not a way to make a late order look on time.

Choosing a new time

New times come from the store's **available promised times**, the same set the ordering flow offers. Pick from those rather than inventing a time: they reflect what that store can actually commit to.

The store's promised times are part of its configuration — see [Store delivery settings](#).

You may be asked to confirm

This is the part worth knowing in advance.

When the change has consequences the system wants you to acknowledge, the first attempt **does not go through**. Instead you are shown the specific warnings and asked to confirm. Confirming re-submits the same change, now with your acknowledgement attached.

What you'll see	What it means
The change applies immediately	No side effects worth flagging
Warnings and a confirmation prompt	Real consequences — read them, then confirm if you still want it
The confirmation has expired	You took too long. Start again; the warnings are re-evaluated against the current state

The confirmation is deliberately not a generic "are you sure?". The warnings name what will actually happen. Read them — that is the whole point of the step, and the state may have moved since you opened the order.

Orders already with a carrier

When the order is with an external carrier, the change goes to the carrier **first**, and only then is it recorded on your side.

That order matters: the carrier call is the irreversible half and the one that can fail. Doing it first means a failure leaves both sides unchanged, rather than leaving your records showing a new time while the carrier's courier is still working to the old one.

If the carrier rejects the change, the promised time is not updated. That is not a bug — it is the system refusing to record a promise the carrier has not accepted.

Before you change it

1. **Has the customer been told?** The promised time is a commitment, and changing it silently just moves where the disappointment lands.
2. **Is the real problem the assignment?** If the order is simply unassigned, assigning it may be the fix — see [Assigning and unassigning a driver](#).
3. **Is this a pattern?** One order is an incident; a store missing its promised times all lunchtime is a configuration or capacity question. See [Store delivery settings](#).

Related pages

- [Logging an incident](#) — to record *why* the time changed
- [The order board](#) — how Pending and Critical are judged against the promised time
- [Store delivery settings](#) — where promised times come from

Logging an incident

“**You need:** *Hub — incidents* to record one. Reading the incident catalogue needs only *Hub — orders and status*.”

An incident records what went wrong on a specific order, so it can be explained later and counted across orders.

The incident catalogue

Incidents are chosen from a catalogue rather than typed free-hand. The catalogue is what makes incidents countable: "how often did this happen last month" only has an answer if everyone picked the same entry for the same situation.

Alongside the catalogue entry you add a **comment**. The entry says *what kind* of problem it was; the comment says what actually happened on this order.

Recording one

1. Open the order — see [Order details and pickup code](#).
2. Choose the incident from the catalogue.
3. Write a comment with the specifics: what happened, what was done, who was involved.
4. Save.

Writing a comment that's worth reading

The comment is read weeks later by someone who wasn't there. What helps:

- **Times.** "Driver arrived 19:40, order not ready until 19:55" beats "order was late".

- **Names or identifiers**, not roles. "Carrier X's courier never arrived" beats "the courier didn't show".
- **What was done about it**. Whether the customer was called, whether a refund was offered.
- **The customer outcome**. Whether they were satisfied.

What doesn't help: "late", "problem", "see phone call".

Incidents are a record, not an action

Recording an incident does not change the order. It does not reassign, cancel, change the promised time or notify anyone automatically. If the order also needs one of those, do it separately:

Also need to...	Page
Move it to another driver	Assigning and unassigning a driver
Change what the customer was promised	Changing the promised time
Stop it entirely	Cancelling an order
Close it as delivered	Forcing a delivery

“ Record the incident **as well as** taking the action, not instead of it. An order that was force-delivered without an incident explaining why is the hardest kind to reconstruct later.

When to record one

Whenever something happened that a person would need explained later: a late delivery, a missing or wrong item, a driver or carrier failure, a customer complaint tied to this order, or any use of cancel or force delivery.

Consistency matters more than volume. An incident type nobody records is indistinguishable from a problem that never happens.

Cancelling an order

“ **You need:** *Hub — cancel*. Typically `hub_manager` only — deliberately not given to `hub_operator` .

Cancelling stops an order entirely. It is one of the two critical Hub actions, alongside [Forcing a delivery](#).

Before you cancel

Check	Why
Is cancelling actually what's wanted?	If the order is just going to be late, changing the promised time may serve the customer better — see Changing the promised time
Has the customer been told?	A cancellation they learn about from a missing dinner is a different conversation from one they agreed to
Is the order already out for delivery?	If the driver has left with it, cancelling here does not recall them
Do you know the reason?	It is required, and it is what anyone reviewing this later will read

Cancelling

1. Open the order.
2. Choose cancel.
3. Enter the **cancellation reason** — required, and worth writing properly.
4. Confirm.

If the order is with a carrier

When an external carrier holds the order, the cancellation covers **both sides together**: the order is cancelled with the carrier and released on your side as one operation, not two.

This matters because the failure mode it prevents is a nasty one. Cancelling only your side would leave the carrier still holding a live order, with a courier on the way to collect something nobody expects them to collect.

“ If the carrier side fails, the cancellation does not silently half-apply. Check the order's state afterwards rather than assuming it went through, and retry rather than cancelling again from a different screen.

Write a real reason

The reason is the only explanation that survives. It is read by whoever handles the customer's follow-up, and by anyone reviewing cancellation rates for that store.

Useful: "Customer called to cancel — ordered twice by mistake, ticket 4471 kept." **Not useful:** "cancel", "error", "customer".

Also log an incident

Cancelling records *that* the order stopped, with your reason. It does not categorise the cancellation into anything countable.

If the cancellation came from something that went wrong — rather than a customer simply changing their mind — log an incident as well, so it shows up in the counts. See [Logging an incident](#).

What cancelling does not do

- It does not recall a driver already out with the order.
- It does not issue a refund. That is handled wherever your refunds are handled.
- It does not notify the customer automatically. Assume they need to be told.

Forcing a delivery

“**You need:** *Hub — force delivery*. Typically `hub_manager` only. This is the most consequential action in the Hub — read the whole page before using it.

Force delivery closes an order as delivered when the normal flow could not close it — most often because the driver's device failed to register the delivery, or an external carrier never reported completion.

What it actually does

It records the order as delivered, with a driver and a distance that **you** supply rather than ones the system observed.

That is the whole point, and the whole risk: you are asserting something the system could not confirm. Everything downstream — delivery counts, driver figures, distance-based reporting — takes your word for it.

Use it only when the delivery really happened

Force delivery is for correcting a **recording** failure, not an operational one.

Situation	Force delivery?
Customer received the order; the app never registered it	☐ Yes — that is exactly this
Carrier delivered but never reported back	☐ Yes
Order is very late and you want it off the board	☐ No — it is still in progress
Order was never delivered	☐ No — cancel it instead
You are not sure whether it was delivered	☐ No — find out first

Forcing an order that was never delivered puts a false delivery into your records permanently. There is no clean undo. When in doubt, confirm with the driver or the customer before acting.

Doing it

1. Open the order — see [Order details and pickup code](#).
2. Choose force delivery.
3. Enter **who delivered it**: your own driver, or the carrier. For an own driver you name the specific person; for a carrier the company is enough, since the courier is theirs.
4. Enter the **estimated distance** for the delivery.
5. Confirm.

The distance matters

The distance you enter is used as the delivery's distance. It feeds driver figures and any distance-based reporting or settlement.

Estimate it honestly from the store to the delivery address. A placeholder — zero, or a round number entered to get past the field — quietly corrupts whatever is calculated from it, and nobody will notice until the numbers are questioned months later.

Always log an incident too

A forced delivery with no incident is the hardest thing to reconstruct later: the record shows a normal delivery, with nothing explaining why a person had to close it by hand.

Log an incident describing why the force was needed — device failure, carrier silence, whatever it was. See [Logging an incident](#).

If you are unsure

Do something reversible instead. Change the promised time, log an incident, or leave the order alone and escalate. Every other Hub action can be worked around afterwards; this one cannot.

Stores and driver alerts

“ **You need:** *Hub — orders and status* for the store view, *Hub — driver alerts* for driver alerts.

Alongside the order board, the Hub shows the state of your stores and raises alerts about drivers.

The store view

A per-store operational view of the stores within your scope: what each one currently has in play, so you can tell at a glance which store needs attention rather than reading the whole order board.

Use it as the entry point during a busy service. Find the store that looks wrong, then drill into its orders.

“ This is the Hub's *operational* view of a store. Changing how a store behaves — delay tolerances, pausing delivery — happens elsewhere; see [Store delivery settings](#) and [Pausing and resuming delivery](#).

Driver alerts

Driver alerts flag drivers whose behaviour needs looking at rather than orders that are late.

The most common is the **pickup delay** alert: a driver has been at the store for too long without leaving with an order that is ready. It fires from the same at-store determination the dispatcher uses for priority — roughly 350 metres from the store. See [Driver classification and priority](#) for what "at store" means and everywhere else it is used.

Others cover drivers who appear stuck or unavailable in ways that will affect assignment if left alone.

Acting on an alert

An alert is a prompt, not an instruction. Work out which of these it is:

What you find	What to do
Driver is genuinely delayed at the store	An operational conversation, not a console action
Driver's device or connection has dropped	Their orders may need reassigning — Assigning and unassigning a driver
Driver has finished but the system doesn't know	The order may need Forcing a delivery — read that page first
The alert is firing too readily across many drivers	The threshold is wrong — Alarm settings

“ A single alert is about one driver. The same alert firing across many drivers at once is about the threshold, or about something systemic at that store. Don't work through fifty alerts one at a time — check the threshold first.

Alerts, alarms, incidents

Three similar-sounding things that are genuinely different:

	What it is	Where
Driver alert	The system flagging a driver's state	This page
Branch alarm	The system flagging a store's state	Branch alarms
Incident	A person recording what went wrong on an order	Logging an incident

The first two are raised automatically and you respond to them. The third is created by you and is a permanent record on one order.

Branch alarms

“**You need:** *Hub — branch alarms* to view. Archiving needs *Hub — archive alarms*, typically `hub_manager`.”

Branch alarms flag that a **store** needs attention — as opposed to driver alerts, which flag a person, and incidents, which you record yourself.

What they tell you

An alarm fires when a store crosses a configured threshold: too much load, too many cancellations, too many orders missing their promised time. What counts as "too much" is configurable per alarm type — see [Alarm settings](#).

An alarm is a signal, not a diagnosis. It tells you where to look, not what is wrong.

Working through them

1. **Look at the store, not the alarm.** Open its orders and see what is actually happening.
2. **Decide whether it is an operational problem or a threshold problem.** One store alarming during a genuine rush is the system working. Every store alarming every lunchtime means the threshold is wrong.
3. **Act on the underlying issue**, if there is one — reassign, pause delivery, change a promised time.
4. **Archive the alarm** once it has been handled.

Archiving

Archiving marks an alarm as dealt with and takes it off the active view. It doesn't delete it: archived alarms remain available for review — see [Archived alarms](#).

Archive when the alarm has been **handled or understood**, not merely seen. An alarm archived without being looked at is worse than one left active, because it now looks resolved to everyone

else.

“ Archiving is a higher permission than viewing, on purpose. Anyone on shift can see what is firing; deciding something has been dealt with is a shift-manager judgement.

When the board is full of alarms

Alarms that fire constantly stop being read. If a store or an alarm type is permanently lit:

- Check the threshold against what that store can realistically do — [Alarm settings](#).
- Check whether it is the same root cause producing many alarms, rather than many problems.
- Consider whether the store's delay tolerances match its reality — [Store delivery settings](#).

Tuning a threshold so it fires when someone should act, and stays quiet otherwise, is worth more than working through a long list.

Related pages

- [Stores and driver alerts](#) — the driver-side equivalent
- [Alarm settings](#) — where thresholds are configured
- [Archived alarms](#) — reviewing what has been handled
- [Logging an incident](#) — recording what went wrong on a specific order

Alarm settings

“ **You need:** *Alarm settings* — view to read, *Alarm settings* — edit to change. The dedicated role is `alarms_config_manager`; `admin` also has it.

Alarm settings decide when an alarm fires. They apply across the whole chain, not per store.

The three families

Type	Watches	Example of what it catches
Order	Individual orders	Orders sitting too long, or past their promised time
Driver	Drivers	A driver at the store too long without leaving — the pickup delay alert
Branch	Stores	A store under too much load, or with too many cancellations

Each family holds several individual alarms. You can browse a family, open one alarm and change its thresholds or turn it off.

Chain-wide, not per store

This is the thing to internalise before changing anything: a threshold applies to **every store**.

Tuning an alarm because one busy city-centre store is noisy also retunes it for the quiet suburban ones — where it may now never fire at all. You are always trading a false-positive problem at one store for a false-negative problem at another.

“ If the real need is genuinely per store, a chain-wide threshold cannot express it. Set the threshold for the majority, and handle the outlier operationally rather than by moving the number until that one store goes quiet.

Changing a threshold

- 1. Open the alarm family — order, driver or branch.
- 2. Open the specific alarm.
- 3. Change the threshold, or switch the alarm off.
- 4. Save.

Changes affect what fires from then on. They do not retroactively clear alarms already raised — clear those by archiving them, see [Branch alarms](#).

Choosing a number

A good threshold fires **when someone should act** and stays quiet otherwise. Both failure modes are real:

Too sensitive	Too lax
Fires constantly	Fires rarely or never
People stop reading alarms	Real problems pass unnoticed
Genuine problems buried in noise	Discovered from customer complaints instead

The second is more dangerous; the first is more common — and it causes the second, because an alarm board nobody reads is functionally an alarm that never fires.

Before changing a number, look at what actually fired over the last week or two. Tuning from a specific bad shift produces a threshold shaped by an outlier.

Turning an alarm off

Switching one off is legitimate when it does not apply to how you operate. It is not a fix for an alarm that fires too often — that is a threshold question, and turning it off removes the signal entirely rather than sharpening it.

Note what you turned off and why. An alarm silently disabled a year ago is indistinguishable from one that never fires.

Related pages

- [Branch alarms](#) — acting on store alarms
- [Stores and driver alerts](#) — acting on driver alerts
- [Archived alarms](#) — the history to tune against

Archived alarms

“**You need:** *Archived alarms — view*. Exporting needs the metrics export permission. `hub_manager` holds the view; chain-wide export is typically `admin`.”

Archiving an alarm doesn't delete it. Archived alarms are kept and can be reviewed later — the history is where you find patterns that no single shift reveals.

Where to find them

Archived alarms live with **Metrics**, not with the live Hub board. That split is deliberate: the Hub is for what needs acting on now, Metrics is for what happened.

What the history is for

Question	How the history answers it
Is this store always like this?	Compare its alarm volume against comparable stores over the same period
Is our threshold sensible?	An alarm archived in seconds every time is firing too readily — see Alarm settings
Did that change help?	Compare the weeks either side of a configuration change
Which alarms does nobody act on?	An alarm type that is always archived immediately, with nothing else happening, is noise

Tuning thresholds from history

This is the main reason the history exists. Before changing a threshold in [Alarm settings](#), look at what that alarm actually did over a representative period — a fortnight, not one bad shift.

Two patterns are worth acting on:

- **Fires constantly, always archived immediately.** Too sensitive. Raise it, or switch it off if it doesn't apply to how you operate.
- **Never fires, yet the problem shows up in customer complaints.** Too lax. Lower it.

Remember thresholds are chain-wide: a change made to quieten one store changes every store.

Exporting

The history can be exported for analysis outside the console. Export is a separate, higher permission than viewing, because an export leaves the console's access controls behind entirely.

“ An exported file is no longer scope-filtered. Whoever opens the spreadsheet sees everything in it, regardless of what they could see in the console. Treat exports accordingly, and prefer sharing a conclusion over the raw file.

Related pages

- [Branch alarms](#) — the live view and archiving
- [Alarm settings](#) — where thresholds are changed
- [Stores and driver alerts](#) — driver-side alerts