The Job Scheduler: work that happens without you

Most of what VeloxFactory does happens because something asked for it: a request comes in, a report is rendered, a label goes to a printer. Some work has no caller, though. A shift report that should be in the supervisor's mailbox at six every morning. The cleanup that keeps the history from growing forever. Work like that runs on a clock, not on a request.

The **Job Scheduler** is where that clock lives. Every job is a record with a timezone and an active flag, and it says when it runs either as a crontab expression or as a single date and time. Schedules are maintained in the frontend or through the API like any other master data. VeloxFactory itself runs one fixed entry: every minute it looks for schedules that are due and hands them to the queue.

Screenshot placeholder: `scheduler.overview.png`
The Job Scheduler overview with type, cron expression, next run, last run and status per schedule.

Two kinds of schedule

Type	What it does
Render Job	Fires a complete render request on a schedule: parameters, data rows, printing, history record and the full mailing segment. Everything a render can do from the API, a schedule can do at four in the morning.
Purge Job	The housekeeping: print tasks, mail tasks, history records, orphaned files and the scheduler's own run log, each with its own retention in days. One schedule, one run, a fixed order.

One Purge Job covers all five kinds of cleanup, so there is a single place to say when housekeeping happens and how long each entity is kept. A fresh instance already has the schedule, deactivated and without a cron expression, so nothing runs until you decide when it should.

Saying when

A schedule accepts full five field crontab syntax, the same thing every operations person already knows. You do not have to write it by hand: the editor has a guided mode with frequency, time and weekday pickers, and an expression mode for everything the guided mode cannot express. Both write into the same field, and whichever you use, the next run times are shown underneath while you type.

Expression	Meaning
`0 6 * * 1-5`	Weekdays at 06:00, the shift report.
`*/15 * * * *`	Every quarter of an hour.
`0 3 1 * *`	The first of every month at 03:00, the monthly statement.
empty	Never fires on its own. The schedule exists and can be started by hand, which is how you test one before switching it on.

Every schedule carries its own timezone, defaulting to the timezone of the instance. That matters for a company that renders for a plant in another country: the expression says 06:00 and the timezone says which 06:00.

> **Daylight saving time:** an expression that points into the hour the clock skips or repeats runs twice in October and not at all in March. For a job that has to happen exactly once a day, pick a time outside 02:00 to 03:00.

Running once, at a given time

Not every job repeats. A price list that has to go out on the day the new catalogue takes effect, a cleanup that belongs to one migration: those are a point in time, not a pattern. A schedule therefore says when it runs either with a cron expression or with a single date and time, never with both.

```
{
```

```
}
```

In the form the field is **Run once at**, with a date and a time picker, next to the switch **Deactivate after the run**. A one-time schedule shows *Once* in the overview instead of an expression, and its date on the detail page and in the next run preview.

Question	Answer
Which clock does the time follow?	The timezone of the schedule, or the timezone of the instance when the schedule has none. A value that carries its own offset, such as `2026-10-25T17:35:00+02:00`, wins over both. The API answers with the time in the schedule's timezone.
Cron and date together?	Refused with 422. To turn a one-time schedule into a recurring one, send `runAt: null` together with the cron expression.
A date in the past?	A date that is being set or changed has to lie in the future. A date that already passed does not block edits to the other fields of the schedule.
What happens after the run?	The date is used up: there is no next run and the schedule does not fire again. That also holds when the slot was discarded after a long outage of the scheduler, or when the worker died in the middle of the run.
And the schedule itself?	With **Deactivate after the run** it sets itself inactive. Without it the schedule stays active but has nothing pending; give it a new date to arm it again.
Does **Run now** use the date up?	No. A manual run is a test, the pending date stays where it is.

A one-time Purge Job counts as an active cleanup for as long as its date is still ahead, so the overview does not warn about missing housekeeping while one is armed.

What happens when a schedule is due

Every minute VeloxFactory collects the schedules whose next run has arrived, moves each one to its following slot and hands it to the queue. The order matters: the next slot is written first, so a schedule can never fire the same slot twice, whatever happens afterwards. A schedule that runs once has no following slot, so its slot is cleared instead of moved on, which is what makes it fire exactly once.

Three things are deliberately not done.

- **Missed slots are not made up.** If the scheduler was stopped for three hours, the slots in between are gone. One late run within the grace window is fired, anything older is recorded as skipped so you can see that it happened.
- **Runs of one schedule never overlap.** A render that takes longer than its own interval simply keeps running; the next run is recorded as skipped rather than queued behind it. Two different schedules do run side by side.
- **A run is not retried.** A second attempt would print a second label and send a second mail. A failed run ends as an error and says why.

Who a render job runs as

A scheduled render is a real render request, so it needs somebody to make it. That somebody is the user who created the schedule, and the schedule carries one of that user's API tokens to run with. Nothing else changes: the same permissions apply, the same audit columns are filled, and the history record shows who and which token, exactly as if the request had come over HTTP.

Two consequences are worth knowing before you build a schedule around a colleague who is about to change roles.

- If the owner is set inactive, or the token is revoked or expires, the run stops before anything is rendered. It is recorded as skipped with the reason, and the failure mail goes out.
- Because a render job acts with its owner's permissions, only the owner and administrators may change its configuration or start it by hand. Everyone with the read permission can see it and read its log.

The token is referenced, never stored. VeloxFactory keeps only the hash of a token, so a schedule points at the token record you pick from a list, and revoking that token stops the schedule the same minute.

API tokens expire after the period configured in `SANCTUM_EXPIRATION`, one year by default. A schedule whose token has lapsed stops with the same error as a revoked one and sends the configured failure mail. Owners of long-running schedules should issue themselves a fresh token before the old one expires.

Screenshot placeholder: `scheduler.render-form.png`
The form of a render job: cron builder, report config, API token, output settings, parameters with the placeholder switch and the mailing block.

The run log

Every run writes a record, whether it worked or not. The log carries the trace id, so a scheduled render can be followed through the history record, the print task and the mail task it produced, and back again.

Status	Means
`Success`	Everything the schedule asked for happened.
`Warning`	The job ran, but something inside it did not: the render worked and the mail did not, a placeholder stayed empty, one purge step failed while the others finished.
`Error`	The job itself failed. The message from the API is on the record.
`Skipped`	The run never started: the previous one was still going, the owner or the token was not usable, or the slot was older than the grace window.
`Running`	In progress right now.

A run also records whether it was fired by the clock or started by hand, so a test run is never mistaken for a scheduled one.

Being told when something breaks

A schedule that quietly stops working is worse than no schedule at all. Every schedule can name a mailer and a list of recipients for its failure mail, which goes out when a run ends as an error or is skipped. A warning does not mail: the render happened, and the detail is already on the history record or the mail task.

These recipients are plain addresses. Placeholders belong to the mail of a document, which is resolved from the render it belongs to, and a run that failed has no render to resolve anything against.

A schedule that fires every minute and fails every minute would otherwise fill a mailbox, so the failure mail is throttled to one per schedule per hour. The window is a setting of the instance, the runs in between count how many were suppressed, and the log still has every one of them.

AI assistants

The MCP server exposes the Job Scheduler the way it exposes the rest of VeloxFactory: schedules and their run log have their own tools, a crontab expression can be validated and its next run times read back before anything is saved, and a schedule can be started once by hand to see what it does. Because report configs, mailers and templates are resolved by name, a schedule is built from what a person said. *Send the shift report to the supervisor every weekday at six* is one request.

The retentions of the Purge Job have their own tool, because the API expects the complete retention map in every write and a single changed value would otherwise be refused.

Permissions

The Job Scheduler has one permission scope, `scheduled-job:*`, with the usual `read`, `create`, `update`, `delete` and `full`. The run log is covered by the same scope.

Two rules go beyond the usual pattern, both for the same reason: a schedule can act with somebody else's rights, or with nobody's.

- Creating a schedule implies being allowed to render, print and mail with it, so `scheduled-job:create` grants what a render needs, the way `scan2print:use` does for the scan station.
- A Purge Job has no owner at all, so whoever switches a cleanup step on has to hold the permission to delete what that step deletes. Turning on the history step needs the right to delete history records.

Deleting a schedule is not limited to its owner. Deleting only stops a schedule, it can never make one act with somebody's permissions.

> **The Purge Job is deletable like any other schedule.** With no active purge schedule, nothing cleans up history records, print tasks, mail tasks or orphaned files any more, and the disk keeps filling. The overview warns while that is the case. To bring the schedule back, use **New** and then **Purge Job** in the frontend, or run `php artisan scheduler:create-purge-job` on the server.

What has to be running

The Job Scheduler depends on the two background processes VeloxFactory runs in production.

Process	Why
Scheduler	The minute tick that finds due schedules. Without it nothing fires at all.
Horizon	The runs themselves. They have their own queue and their own worker, so a scheduled render that takes minutes never blocks a thumbnail or a mail.

Both belong under Supervisor in production. The Administration chapter has the details.

Revision #6
Created 2026-09-21 08:41:49 UTC by Benjamin Fischer
Updated 2026-09-23 05:02:23 UTC by Benjamin Fischer