Render Jobs: a render request with a clock attached

A **Render Job** is a render request that VeloxFactory keeps and sends to itself. Whatever you can put into a call to the render endpoint goes into the schedule: parameters, data rows, the printer, the history record, the complete mailing segment. Nothing about rendering is special-cased for schedules, which is the point. A request you have tested once from the API behaves exactly the same at four in the morning.

Screenshot placeholder: `scheduler.render-parameters.png`
Parameters and data rows of a render job, with the placeholder switch active on a date parameter.

What you configure

Part	What it decides
Report Config	Which report is rendered. Exchangeable later in the frontend and through the API, and the payload is revalidated against the report it now points at.
API token	Which of the owner's tokens the run authenticates with. Only tokens of the owner are accepted, and only ones that are neither revoked nor expired.
Output	What the render produces, and whether a history record is written.
Printing	Printer, number of copies and broadcast id, exactly as on the generate page.
Parameters and data rows	The values the report is rendered with, entered in the same table as on the generate page, here with placeholders allowed.

Part	What it decides
Mailing	The same block as everywhere else: mailer, template, recipients, attachments, background sending.

The **type** and the **owner** are the two things a schedule keeps for its whole life. A render job stays a render job, and it stays the user who created it, because that is whose permissions every run acts with.

> ℹ **A schedule runs even when its owner cannot use the API.** The scheduler calls the render internally and does not pass the API middleware, so `global:use-api` is not required for a schedule to fire. What does stop a run: a revoked or expired token, a token that no longer belongs to the owner, a deactivated owner, or a missing resource permission.

Output types

A schedule offers three of the four output types.

Type	Use it when
`url`	The usual choice. The PDF lands on the history disk, stays downloadable from the history record and is what a mail attaches.
`base64`	Same file, plus the document inside the history record. Comfortable, and noticeably larger per run.
`none`	Printing only. A print task is mandatory with it, the same rule the API applies.

`preview` is refused. A preview render deletes its own file again, so unattended it would produce nothing at all.

> `none` **and mailing do not go well together.** Output type `none` produces no PDF, so a mailing that wants to attach one gets a mail without its attachment. Saving is not refused, the form says so as a note, and the honest combinations are `none` with `includePdf: false`, or `url` .

Placeholders in parameters

A schedule that renders yesterday's figures every morning cannot have a fixed date in its parameters. Parameters and data rows therefore accept placeholders, resolved fresh at the start of every run.

Only those placeholders that exist *before* a render are offered here, so no `[data.first.*]` and no `[report.*]`: those describe a render that has not happened yet.

Placeholder	Resolves to
`[now.date]`, `[now.dateTime]`, `[now.time]`	The moment the run starts.
`[now.year]`, `[now.month]`, `[now.weekNumber]`, `[now.quarter]`	Period markers, for reports that select by week, month or quarter.
`[traceId]`	The trace id of this run, so the document can carry the id the log shows.
`[uuid]`	A fresh UUID, the same value everywhere it appears in one run and a new one in the next.
`[user.name]`, `[token.name]`	The owner of the schedule and the token it runs with.
`[env.hostname]`, `[appName]`, `[appUrl]`	Which instance produced the document.

The Placeholders button above the parameter table lists all of them with a description and an example, and copies a token with one click. `GET /scheduled-job/placeholders` returns the same catalogue.

> **Typed fields need the switch.** A date parameter is a date picker and a number parameter is a number field, and neither accepts `[now.date]` as text. The small button next to the field turns it into a text field for exactly that reason, and back again. Boolean parameters have no switch: a toggle has no text state, so they take no placeholders.

In the mailing block placeholders work as they do everywhere, and there the full catalogue applies: recipients, contact name and attachment names are resolved *after* the render, so `[data.first.customerEmail]` addresses the rows the report actually fetched.

Checked when you save, not at four in the morning

A schedule that cannot possibly succeed is refused while you are still looking at the form.

- Every required parameter of the report has a value or a resource override, named individually if not.
- The owner holds the permissions the run will use: reading report configs, creating print tasks when a printer is configured, creating mail tasks when there is a mailing block.
- Mailer and mail template of the mailing block exist and are active.
- Every resource override path points inside the resources folder of the instance, the same check the render applies.
- The token belongs to the owner and is neither revoked nor expired.
- The cron expression is valid five field syntax, and a date for a single run lies in the future while you are setting or changing it.
- No `traceId` in the payload: every run makes its own, and a stored one would let the first run succeed and every one after it fail.

From the API

The schedule itself is ordinary master data. Its `payload` is the body you would have sent to the render endpoint.

```
{
                                                                    "name":

                                                  "cron":               "0

                                        "parameters":     {     "P_DATE":     "[now.date]",
```

```
        }
    }
}
```

`reportConfig`, `errorMailer` and the mailer and template inside the payload accept either the numeric ID or the unique name, the same way the render endpoint does.

A job that is meant to happen once carries `runAt` in place of `cron`, and says with `deactivateAfterRun` whether it switches itself off afterwards:

```
{



}
```

Both fields come back on every read, `runAt` in the timezone of the schedule. Sending `cron` and `runAt` together is refused with 422, and `runAt: null` together with an expression turns the schedule back into a recurring one.

Endpoint	Does
`GET /scheduled-job`	Lists schedules, filterable by `type` and `isActive`.
`GET /scheduled-job/{id}`	A single schedule with its payload, its owner and its token.
`POST /scheduled-job`	Creates one. The caller becomes the owner.
`PATCH /scheduled-job/{id}`	Changes it. Type and owner are refused, everything else you leave out keeps its stored value.
`DELETE /scheduled-job/{id}`	Deletes the schedule and its run log.
`POST /scheduled-job/{id}/run`	Queues a run right now, with the owner's identity, not the caller's. A pending single run stays pending.
`POST /scheduled-job/preview-cron`	Validates an expression and answers with the next run times in a timezone.
`GET /scheduled-job/placeholders`	The catalogue a schedule may use, for building your own form.
`GET /scheduled-job-run`	The run log, filterable by schedule and status.
`GET /scheduled-job-run/{id}`	One run with its trace id, its result summary and its error message.

Endpoint	Does
`DELETE /scheduled-job-run/{id}`	Deletes one run record. What the run produced is untouched.

Changing or starting a render schedule is limited to its owner and to administrators, because the run acts with the owner's permissions. Reading it, and reading its log, needs `scheduled-job:read` like any other lookup.

Trying it before you trust it

Leave the cron expression empty, save, and press **Run now**. The schedule never fires by itself, the run is recorded as a manual one, and you get the same result you would get at six in the morning: a history record, a print task if you configured a printer, a mail task if you configured mailing, and a line in the run log with the trace id that ties them together.

A schedule that is waiting for its single date can be tested the same way. **Run now** is a test run, not the appointment, so the date stays where it is.

When it does what you want, add the expression and switch the schedule on.

Revision #4
Created 2026-09-21 08:46:05 UTC by Benjamin Fischer
Updated 2026-09-23 05:00:41 UTC by Benjamin Fischer