The concept of Report History Records

Every render request tells VeloxFactory what to produce. A `ReportHistoryRecord` remembers exactly what was asked for, what came back, and what the result looked like, permanently, until you decide otherwise. It is the foundation for traceability, debugging, and on-demand reprinting in VeloxFactory.

Screenshot placeholder: `report-history-record.index.filtered.png`
Report History Record overview with filters applied and status badges.

What Gets Stored

A `ReportHistoryRecord` is created at render time when `createHistoryRecord: true` is set in the request, or automatically when a print task is dispatched. It captures a complete snapshot of the rendering event:

Field	Description
`traceId`	Unique identifier shared across the render request, the history record, and any linked print task. Used to correlate events in logs and across systems.
`reportConfig`	Reference to the `ReportConfig` that was rendered.
`outputType`	The output type used: `base64`, `url`, or `none`.
`apiPayload`	The complete render request body: parameters, data, flags, everything sent to the render endpoint. Stored as JSON.
`apiResponse`	The complete API response returned by VeloxFactory, including any errors and, unless the render was laconic, the input it actually worked with. Stored as JSON.
`reportPdf`	The rendered PDF, Base64-encoded. Present on successful renders; `null` on failure.
`reportPdfFileName`	The UUID-based filename assigned to the rendered PDF.
`reportExcelFileName`	Filename of the xlsx export, present when a mailing asked for one. The file hangs on the record, is downloadable from it and is deleted with it.

Field	Description
`reportThumbnail`	A thumbnail image of the first page of the rendered PDF. Generated asynchronously in the background after the record is created.
`status`	Automatically calculated from the stored response. See below.
`reportPrintTasks`	Every print job dispatched from this record, with printer, copies and status. A record can hold any number of them.
`reportMailTasks`	Every mail sent from this record, with recipients, subject, the rendered body and the attachment names.
`audit`	Who created the record and when, who last updated it, and which API token was used in either case. VeloxFactory derives `creationMethod` and `updateMethod` from it, `Frontend` or `API`, depending on whether a token was present on the request.

The audit block is what turns a history record into an answer to "who printed this". A render triggered by an integration carries the token it was made with, a render triggered in the browser carries the user.

How It Comes Back from the API

The record is stored in full, but a response only carries what you ask for. Four query parameters control this, and they matter, a history record with an embedded PDF is a large object:

Parameter	Default	Effect
`withApiPayloads`	`true`	Includes `apiPayloadBase64` and `apiResponseBase64`. Both are Base64-encoded, so the stored JSON survives transport unchanged.
`withMedia`	`false`	Adds the `media` block with `reportPdfBase64`, `reportPdfFileName` and `thumbnailBase64`.
`withRelations`	`false`	Adds the full `reportConfig`, plus `reportPrintTasks` and `reportMailTasks`.
`withAudit`	`false`	Adds the `audit` block described above.

Without any of them a record comes back as a compact summary: ID, trace ID, output type, status and the payloads. That is the right shape for a list view, and it is what the frontend's history overview uses.

Status

The status of a `ReportHistoryRecord` is calculated automatically every time the record is saved, based on the content of the stored API response:

Status	Meaning
Ok	No errors in the response and a PDF was produced. The render completed successfully.
Error	The response contains one or more errors. The render failed, the error messages are stored in the API response payload.
Render Fail	No errors in the response, but no PDF was produced either. An edge case indicating something unexpected occurred during rendering.
Unknown	Status could not be determined from the stored response.

History records are created for both successful and failed renders. A failed render still produces a complete record, including the error messages, which is often more useful than a successful one when something goes wrong.

The Thumbnail

When a history record is created, VeloxFactory dispatches a background job that converts the first page of the rendered PDF into a thumbnail image using `pdftoppm` (part of `poppler-utils`). The thumbnail is stored on the record and displayed in the history list and the report card grid, giving you an immediate visual of what was produced without opening the PDF.

> ℹ **Thumbnail generation runs asynchronously.** The history record is available immediately after rendering; the thumbnail appears once the background job has completed. This requires the Laravel queue worker (Supervisor) to be running. If the queue is down, thumbnails will not be generated until it is back up.

Traceability and Debugging

The most valuable aspect of a history record is not the PDF, it is the payload. Every record stores the exact request that triggered the render and the exact response that came back. This means you can answer the following questions at any point in the future, without touching the calling application:

- What parameters were passed to this render?
- What data was submitted, and which rows the report was actually filled with?
- Was the render triggered via the frontend or the API?
- What did VeloxFactory return, and did it succeed?
- If it failed, what was the exact error message?

The `traceId` ties everything together. It is present on the history record, on any linked print task, and in the server logs. When something goes wrong in production and you have a `traceId`, you can pull the history record and reconstruct the entire event in seconds.

Screenshot placeholder: `report-history-record.show.png`
Report History Record detail page with status, payload, PDF preview and the actions.

Reprinting from a History Record

A successful history record holds the rendered PDF. That PDF can be dispatched to a printer at any time, without re-rendering the report, using the dedicated print endpoint:

```
POST /api/v1/report-history-record/{id}/print
```

```
{
  "printerName": "WarehousePrinter01",
  "numberOfCopies": 1
}
```

VeloxFactory creates a new `ReportPrintTask` from the stored PDF, assigns it a derived trace ID (the original trace ID with a short random suffix), and dispatches it to the print service. The original history record is linked to the new print task.

This is useful in several scenarios: a print job failed and needs to be retried, a physical document was lost and needs to be reprinted, or a record needs to be dispatched to a different printer than the one originally used.

> **i Reprinting uses the stored PDF, it does not re-render the report.** The document produced is identical to the original. If the report template or its data has changed since the original render, those changes are not reflected in the reprint.

This endpoint is also available directly from the frontend, the history record detail view has a **Print** button that opens a modal to enter the printer name and number of copies.

Screenshot placeholder: `report-history-record.print.modal.png`
Print dialog of a Report History Record with the printer picker and the copies field.

Mailing from a History Record

The same idea applies to mail. A stored rendering can be sent to anyone at any time, without re-rendering:

```
POST /api/v1/report-history-record/{id}/mail
```

```
{
  "mailer": "Office SMTP",
  "mailTemplate": "Report Delivery",
  "to": ["disposition@example.com"],
  "includePdf": true,
  "includeExcel": false,
  "sendAsync": false
}
```

VeloxFactory creates a new `ReportMailTask` from the stored PDF, gives it a derived trace ID, and sends the mail through the chosen mailer. Recipients, contact name and attachment names accept the same placeholders as a mailing in the render request, resolved against the parameters and rows stored on the record.

If `includeExcel` is set and the record does not have an xlsx yet, VeloxFactory builds one on the fly and attaches it to the record. The export uses the rows the report was actually filled with, which includes the rows an SQL adapter fetched at render time: they travel back in the render response and are stored on the record, so a report that gets its data from a live connection exports just as well as one whose rows came in the request.

> ⚠ **One case has nothing to export.** A render made with `laconicResponse: true` stores a response without its input segment. If that request carried no `data` of its own either, the record holds no rows, and a later export answers with a clear error instead of an empty sheet.

> Render with the mailing segment, or without `laconicResponse`, when an xlsx may be wanted afterwards.

In the frontend the detail view has a **Mail** button next to **Print**, opening a dialog with mailer, template, recipients, attachments and the background switch. Every mail sent from this record is listed on the record itself, and in the Mail Queue.

Screenshot placeholder: `mailing.history-record-modal.png`
Mail dialog of a Report History Record with mailer, template, recipients, attachments and the background switch.

> **Mailing uses the stored PDF too.** The document that arrives is identical to the original rendering. The full reference is on Report Mailing.

Retention and Deletion

Automatic Purging

History records are cleaned up by the **Purge Job**, the consolidated cleanup schedule of the Job Scheduler. How long a record is kept is a retention on that schedule, editable in the frontend or through the API; an empty value switches the step off entirely. The `PURGE_HISTORY_DAYS` environment variable supplies the value the schedule is created with, nothing more.

Deletion Constraints

A `ReportHistoryRecord` cannot be deleted while any `ReportPrintTask` or `ReportMailTask` still references it. The linked tasks must be removed first. VeloxFactory provides a dedicated endpoint for the print tasks:

```
DELETE /api/v1/report-history-record/{id}/delete-all-report-print-tasks
```

This removes all print tasks associated with the record in one call. Mail tasks have their own retention on the Purge Job, which runs before the history records for exactly this reason, and they can be deleted individually from the Mail Queue. Once no task references the record any more, the history record itself can be deleted, which also removes its PDF and its xlsx from disk.

Impact on ReportConfig Deletion

A `ReportConfig` cannot be deleted while any history records reference it. This is a deliberate constraint: the history exists as a permanent trace of what was rendered using that configuration. To remove a report configuration entirely, its history records and their linked print tasks must be cleared first, either manually or by waiting for the automatic purge to run.

Revision #13
Created 2026-05-10 10:44:49 UTC by Benjamin Fischer
Updated 2026-09-21 12:29:10 UTC by Benjamin Fischer