Rendering reports in VeloxFactory

Turning a Report Config into a document: from the frontend, from the API, and what every render leaves behind.

Rendering from the frontend

Every report configuration in VeloxFactory has a built-in **Generate PDF** page: a small workstation for one report template that renders it directly from the browser, without writing a single line of code or touching the API. It is the fastest way to produce a PDF, test a configuration, or trigger a print job on demand.

The page follows one principle: **settings once, then only the data.** How the output should be produced, preview or print, which printer, how many copies, whether a history record is kept, is decided once per report and then stays out of the way. From there, every render is nothing but filling in the values that actually change and pressing one button.

Screenshot placeholder: `generate-pdf.page.png`
Generate PDF page with the header row (report name, settings summary, Output Settings button), parameter table and the action row.

Opening the Generate PDF Page

There are two ways to reach the page. In the report configuration overview, each row has an action menu that contains **Generate PDF**, taking you straight to the render page. Alternatively, open a report configuration and use **Generate PDF** from within the report view.

The header row shows the report name and, as a compact summary line, the active data adapter and the stored output settings: mode, printer, copies, broadcast ID and whether a history record is created. The data adapter is either the assigned `ReportConnectionConfig` (including driver and database) or **dyn. Array** when no SQL connection is configured, and it decides which input sections appear further down.

Screenshot placeholder: `report-config.index.action-menu.png`
Report configuration overview, opened action menu with Generate PDF and Download.

Output Settings

The first time you open the page for a report, the **Output Settings** dialog opens by itself, because VeloxFactory does not yet know what you want to do with the result. Afterwards it only opens when you click the **Output Settings** button in the header.

Setting	Default	Description
Mode	Preview and download only	Decides what happens with the rendered PDF, see the table below.
Printer	-	Printer picker over the active printer master data, free text is allowed as well. Required in both printing modes.
Copies	1	Copies passed to the print service, between 1 and 100. Does not trigger multiple renders.
Broadcast ID	-	WebSocket channel of the print service. Filled in automatically when the chosen printer has one stored.
Create History Record	**On**	Saves request, response, PDF and thumbnail of every render to the report history.
Send results via Mail	Off	Mails every render of this report, see Mailing the result below.

The three modes:

Mode	Preview	Print task
Preview and download only	yes	no
Print only, no preview	no	yes
Preview and print	yes	yes

In *Preview and download only* the printer and broadcast fields are hidden, there is nothing to print to. In the two printing modes a printer is mandatory: saving without one fails and the dialog reopens with everything you entered still in place.

Settings are kept **per report configuration in your session**. Two report templates therefore keep independent settings, they survive leaving the page and coming back, and they end when your session does. **Clear settings** at the bottom of the dialog removes them, the dialog then opens by itself on the next visit.

> ℹ **The settings are yours, not the report's.** They live in your session, so a colleague configuring the same report for a different printer never affects your workstation.

Screenshot placeholder: `generate-pdf.settings-modal.png`
Output Settings dialog with mode dropdown, printer picker, copies, broadcast ID, history toggle, the mailing block and the Clear settings button.

Mailing the Result

Send results via Mail turns every render of this report into a mail as well. While the switch is on, the dialog shows the mailing fields:

Field	Description
Mailer / Mail Template	The SMTP account and the template the mail is built from. Only active records are offered.
To, CC, BCC	Comma separated address lists. Placeholders are allowed, an address containing one is checked after the render, not in the dialog.
Contact Name	Free text for a personal greeting, available in the template as `[contactName]`.
Attach PDF Report	On by default, with an optional name the attachment carries in the mail.
Attach Excel Report	Attaches an xlsx built from the rendered rows, with an optional attachment name and a switch for the parameter block above the table.
Send in background	Hands the mail to the queue instead of waiting for the SMTP server.

The values stay in the form even while the switch is off, so turning mailing off for one render loses nothing. After the render the notification says what happened to the mail: sent, queued, or a warning with the reason. A mail that fails never turns a successful render into an error.

> ℹ **The mailing block only appears when you can use it.** It needs the permission to create mail tasks plus at least one active mailer and one active mail template. The same block sits in the Scan2Print configuration and on a render job in the Job Scheduler. Details on Report Mailing.

Report Parameters

If the report defines parameters, a parameter input table is shown. Each parameter gets its own typed input field, derived automatically from the Java class declared in the `.jrxml`:

- A `java.lang.String` parameter becomes a text field.
- A `java.sql.Date` parameter becomes a date picker.
- A `java.lang.Integer` parameter becomes a number input with integer constraints.
- A `java.lang.Boolean` parameter becomes a toggle switch.
- And so on for all supported types.

Parameters marked as **required** in the report configuration must be filled in before the form can be submitted. Optional parameters can be left empty, VeloxFactory silently drops empty parameter values and does not include them in the render request.

Report Lines - Manual Data Entry

The **Report Lines** section only appears when the report has **no SQL connection assigned**, i.e. when the data adapter is `dyn. Array`. In this case, VeloxFactory has no database to query, so the detail band data must be entered manually in the browser.

The section shows a table with one column per field defined in the report. Each cell contains a typed input matching the field's data type. You fill in one row of values per data record you want to appear in the report.

To add more rows, use the **Add Row** button, it clones the input row and appends a new empty one. Individual rows (except the first) can be removed with the delete button on the right. Empty fields are not transferred to the render request.

> ℹ **When a SQL connection is assigned, the Report Lines section is hidden.** VeloxFactory fetches the data automatically from the database using the configured query, no manual input needed.

Less typing: prefill, example values, clear

Three small helpers remove most of the repetitive work when the same report is rendered again and again:

- **Prefill from the last run.** Parameters and report lines come back filled with the values of your previous render in this session. The values are stored on every submit, successful or not, so a failed render can be corrected instead of retyped.
- **Fill example values.** Fills every parameter and the first line row with the example values stored in the report configuration. This is the frontend counterpart of the API's `useExampleValues`, useful for a quick smoke test of a new template.
- **Clear inputs.** Empties all parameters and removes every line row except the first one.

Resources

If the report has image resources, a collapsible **Resources** section is available on the page. It shows a preview thumbnail of each resource file, its parameter name, and its file name. Resources are handled automatically at render time, you do not interact with them during rendering. The section is informational only, confirming which image files are currently assigned.

Generating

Once the values are in place, click **Generate**. There is no confirmation dialog: the stored output settings already say what is supposed to happen with the result. The request is processed synchronously, the page waits for the result and displays it immediately.

The Result

Success

In the two modes with a preview, the page shows a green confirmation banner, a **Download PDF** button and the generated PDF as an inline preview, sized to the report's actual page dimensions. If a history record was created, the download button serves the stored file, otherwise the PDF is offered straight from the response.

In **Print only**, no preview is rendered at all. The banner then reads *Sent to <printer> (<n>x)*, which is the whole point of that mode: a station that prints and moves on.

If a history record was created, a **View History** button links to the new `ReportHistoryRecord`. If a print task was dispatched, a **View Print Task** button links to the `ReportPrintTask` where you can follow its status.

> ⚠ **Reloading the page renders the report again.** The result lives in the response, so a browser reload repeats the render, including a new print task if the mode creates one.

Screenshot placeholder: `generate-pdf.result.png`
Generate PDF page after a render, with the result block and the inline PDF.

Errors

If the render fails, the page shows a red error banner listing all error messages returned by VeloxFactory, and everything you entered stays on screen. Common causes are missing required parameters, a SQL query that returns no rows, or a resource file that was removed after the configuration was last saved.

If a history record was created before the error occurred, the failed attempt is recorded including the error details, which is useful for diagnosing what went wrong.

Rendering with our powerful API

Everything the Generate PDF page does in the browser, the API does programmatically, with more control, lower overhead, and the same rendering engine underneath. A single `POST` request renders a report, optionally logs the result, and optionally dispatches a print job, all in one call.

Screenshot placeholder: `api-docs.png`
The interactive API reference at /docs, generated by Scribe.

The Render Endpoint

```
POST                                    /api/v1/report-config/{id}/render
```

The `{id}` segment accepts either the numeric ID of the `ReportConfig` or its **report name**, the name set in Jaspersoft Studio and stored in VeloxFactory. Both of these are equivalent:

```
POST                                    /api/v1/report-config/1/render
POST                                    /api/v1/report-config/A5_KanBan/render
```

Using the report name is convenient for integrations: it stays stable even if the database record is recreated, and it makes the request self-documenting.

Request Body

Field	Type	Required	Description
`outputType`	string	✓	Output format: `base64`, `url`, or `none`. See below.

Field	Type	Required	Description
`parameters`	object		Key-value map of parameter names to values. Required parameters must be present or the request is rejected.
`resourceOverrides`	object		Per-render override for `P_RESOURCE_*` image parameters, keyed by parameter name. See Dynamic Resource Overrides below.
`data`	array		Array of field objects, one per detail band row. Each object's keys must match the report's field names. Only needed when no SQL connection is configured. Can include per-row images, see Per-Row Images via Data Fields below.
`createHistoryRecord`	boolean	✓	Whether to create a `ReportHistoryRecord` for this render. Stores the full request, response, and rendered PDF.
`createPrintTask`	boolean	✓	Whether to dispatch the rendered PDF to the print service.
`printerName`	string	if print task	Target printer name. Required when `createPrintTask` is `true`.
`numberOfCopies`	integer		Number of copies passed to the print service. Defaults to `1`. VeloxFactory always renders once, the print service handles duplication.
`broadcastId`	string		WebSocket channel ID. If provided, VeloxFactory broadcasts a `ReportPrintTaskCreated` event when the print task is created. Omit to rely on polling.

Field	Type	Required	Description
`useExampleValues`	boolean		Use the stored example values instead of supplying `parameters` and `data`. Useful for testing. API-only - not available in the frontend. See below.
`laconicResponse`	boolean		Return only the essential output fields instead of the full response. Reduces payload size significantly for high-frequency rendering. See below.
`traceId`	string		Custom trace identifier for this request. Auto-generated (UUID) if not provided. Must be unique across all history records if supplied.
`mailing`	object		Send the rendering by mail after a successful render. See Mailing the rendering below.

Dynamic Resource Overrides

Every `P_RESOURCE_*` image parameter (a logo, a product photo, a line-art reference) normally resolves to whichever file is uploaded or linked on the `ReportConfig` itself. `resourceOverrides` lets a single render request swap that image out, without touching the report configuration, useful for per-customer branding, per-item product photos, or any other case where the image genuinely varies from call to call.

```
POST                                                    /api/v1/report-config/A4_AssemblyBOM/render

{

                    "parameters":           {           "P_PROJECT_NUMBER":           "PRJ-2026

              "P_RESOURCE_ASSEMBLY_IMAGE":    {    "path":    "https://cdn.example.com/parts/4471.
  },
```

```
}
```

`resourceOverrides` is an object keyed by resource parameter name. Each entry supports exactly one of two modes:

Key	Type	Description
`path`	string	A path to a readable file **inside the resources folder** of the instance, or an `https://` URL. Remote URLs are fetched directly by the render engine at render time. A path that leaves the resources folder and any scheme other than `https` are refused.
`base64`	string	Base64-encoded image bytes. VeloxFactory decodes them, writes a temporary file for the duration of the render, and deletes it immediately afterward.
`fileName`	string	Optional, used only alongside `base64` to infer the file extension.

Base64 example:

```
{
    }
  },
}
```

> ⚠ **A resource must still be linked or uploaded on the `ReportConfig` before it can be overridden.** `resourceOverrides` replaces the resolved path for a single render, it does not exempt a `P_RESOURCE_*` parameter from needing a default resource in place. Rendering still fails with the usual `"Not all resources for this report have been uploaded yet!"` error if no file or `CommonReportResource` is linked at all.

> **i Do not set the same parameter in both `parameters` and `resourceOverrides`.** VeloxFactory rejects the request with a `422` if a key appears in both, to avoid ambiguous precedence.

```
{

  "errors": ["Parameter 'P_RESOURCE_LOGO' is set in both 'parameters' and 'resourceOverrides'
-                          use                          only                          one."],

}
```

Other validation errors follow the same pattern: an unknown resource parameter name, a `path` that leaves the resources folder or cannot be read, a URL that is not `https://`, or invalid `base64` all return a descriptive `422` rather than failing deep inside the render engine.

Per-Row Images via Data Fields

`resourceOverrides` covers one image per report-level `P_RESOURCE_*` parameter, the same picture used across the whole render. Some reports need the opposite: a different image for every row of the detail band, for example a product photo per line item in a parts list or an order. That case does not need `resourceOverrides` at all, it works through the `data` array directly.

An `<image>` element in a `.jrxml` is not limited to `$P{...}` parameter expressions, it accepts a field expression just like any text field:

```
<image>
                        <reportElement      x="0"      y="24"      width="120"      height="10
    <imageExpression><![CDATA[$F{partPhoto}]]></imageExpression>
</image>
```

With that in place, every object in the render request's `data` array can carry its own `partPhoto` value, resolved independently per row, using the same three delivery styles as `resourceOverrides`:

```
POST                                                /api/v1/report-config/PartsList/render

{
```

```
            {   "partNumber":   "4471-A",   "partPhoto":   "https://cdn.example.com/parts/4471.jpg
               {    "partNumber":    "4471-B",    "partPhoto":    "/var/www/resources/parts/4471b.
    { "partNumber": "4471-C", "partPhoto": "..."
}
  ],



}
```

Field value	Behaviour
Local path	Read directly from the filesystem by the render engine.
`http(s)://` URL	Fetched directly by the render engine at render time, one request per row.
`...`	Decoded inline, no temporary file involved.
`...`	Same as PNG. Note the MIME token must be exactly `image/jpg`.

⚠ **`...` is not recognized.** The render engine only matches the literal prefixes `` and `` - the far more common `image/jpeg` MIME token is not one of them and the image silently fails to render. Always encode JPEG uploads with the `image/jpg` token in the data URI, regardless of what the source tool actually calls the file.

i No request-level syntax needed. Unlike `resourceOverrides`, per-row images are just regular field values, there is no validation, no linked-resource requirement, no restriction on the location, and no separate object in the request body. Whatever the field resolves to at render time is handed straight to the render engine.

Output Types

The `outputType` field controls how, or whether, the rendered PDF is returned.

`base64`: The PDF is Base64-encoded and returned inline in `output.reportPdfBase64`. No file is written to disk. This is the most common choice for integrations that process the PDF immediately.

`url`: The PDF is saved to the VeloxFactory history storage and a URL pointing to that file is returned in `output.reportUrl`. Useful when the calling application needs to hand off a link rather than handle raw bytes.

`none`: No PDF data is returned at all. Valid only when `createPrintTask` is `true`, the PDF is rendered internally and handed to the print service without being exposed in the response. Use this when the response payload is irrelevant and you only care about getting the document to the printer.

> ⚠ **`outputType: none` requires `createPrintTask: true`.** Requesting output type `none` without a print task is rejected with a validation error, there would be nothing to do with the rendered PDF. A `mailing` segment does not replace the print task here, and a mail on a render that produced no PDF goes out without its attachment.

---

useExampleValues - API-only Testing Mode

When `useExampleValues: true` is set, VeloxFactory ignores any `parameters` and `data` in the request body and instead uses the example values stored on the `ReportConfig`. This is the same data used to generate the report preview in the frontend.

It is a convenient way to verify that a report renders correctly after configuration changes, no test data needs to be assembled:

```
POST                                                    /api/v1/report-config/A5_KanBan/render

{




}
```

> ℹ **`useExampleValues` is an API-only feature.** The Generate PDF page in the browser always requires parameters and data to be entered manually. For frontend testing, use the example values from the report configuration edit page.

---

laconicResponse - Minimal Output

By default, a successful render response includes the full `ReportConfig` record, the input parameters and data echoed back, and any linked `ReportHistoryRecord` and `ReportPrintTask`. For many production integrations, this detail is unnecessary, the caller only needs the PDF.

Setting `laconicResponse: true` strips the response down to the essentials: just the `traceId` and the `output` block. Everything else (`input`, `reportConfig`, `reportHistoryRecord`, `reportPrintTask`) is omitted.

The two response shapes are shown in detail in the Response Structure section below.

> **i The laconic mode also suppresses `reportMeta` in error responses.** If a render fails in laconic mode, the error response contains only the error messages, the field and parameter metadata is not included.

A Complete Request

Here is a full render request for a KanBan label, dynamic array data, a parameter, history logging enabled, print task dispatched via WebSocket:

```
POST                                                    /api/v1/report-config/A5_KanBan/render

{




  },

    {

                                    "description":                  "Packing      Carton




    }
  ],
```

```
}
```

Response Structure

Full Response

The full response (default, `laconicResponse: false` or omitted) includes the rendered output, the echoed input, the full `ReportConfig` snapshot, and any created `ReportHistoryRecord` and `ReportPrintTask`:

```
{
                                        "parameters":     {
     {
                                      "description":          "Packi
     }
    ]
  },
```

```
  },

    ...
  },

  },
  }
},
    "links": { "first": null, "last": null, "prev": null, "next": null
  }
```

```
  },

}
```

Laconic Response

With `laconicResponse: true`, the response contains only what is needed to retrieve the PDF:

```
{

    }
  },
      "links":   {   "first":   null,   "last":   null,   "prev":   null,   "next":   null

    }
  },

}
```

The `traceId` is always included, it links this render to any created history record or print task, making it useful for cross-referencing even in laconic mode.

The envelope around `data` is the one every endpoint uses: a render reports a single page in `meta.pagination` and carries no page links. See Response Structure on Meet the API.

Mailing the Rendering

A render request can carry an optional `mailing` segment. Once the render succeeded, VeloxFactory sends an HTML mail through a configured mailer, using a configured mail template, with the PDF attached and, on request, an xlsx export of the same data.

```
POST                                                    /api/v1/report-config/DeliveryNote/render

{

                              "parameters":               {               "P_ORDER_NO":

                                                                                                    }
}
```

`mailer` and `mailTemplate` take the ID or the unique name of the master data record. Recipients, contact name and attachment names accept the same placeholders as the mail template, which is what makes the segment useful for reports fed by an SQL adapter: `[data.first.<column>]` addresses the rows the query actually fetched, so the recipient is a result of the render rather than an input to it.

The created mail task is returned in `reportMailTask`, next to `reportPrintTask`.

> ℹ **A failed mailing never fails the render.** Whatever goes wrong with the mail, the render response stays a success and the detail lands in `meta.mailing`. The one combination refused up front is `outputType: "preview"` together with `mailing`, a preview render deletes its own file immediately and has nothing to attach.

The full reference, including every field, the placeholder rules, the rate limits and how to mail a rendering that already exists, is on Report Mailing.

Running It on a Schedule

Everything on this page describes a request somebody makes. The same body can be handed to the Job Scheduler instead, which then makes the request itself, on a crontab expression, as the user who created the schedule and with one of that user's API tokens.

The payload of a render job is this body verbatim, with two differences: `traceId` is left out, because every run generates its own, and `outputType` is limited to `base64`, `url` and `none`. Parameters and data rows additionally accept placeholders such as `[now.date]` or `[uuid]`, resolved at the start of each run.

> ℹ See Render Jobs for the schedule around it, and The Job Scheduler for how runs are fired and logged.

Errors

Validation Errors - HTTP 422

Missing required fields, an invalid `outputType` value, or a missing `printerName` when a print task is requested all produce a `422` response with an `errors` array describing the violations.

Required parameters that are not present in the request also return a `422`, one error message per missing parameter:

```
{
                                                                    "Parameter
                                                                        "Parameter
  ],
                                  "meta":                {                "traceId":
}
```


Render Errors - HTTP 400

If the request passes validation but the renderer itself fails, for example an empty data array for a report with a detail band, an SQL query that returns no rows, a type mismatch between field values and declared Java types, or a broken SQL query, the response comes back with HTTP `400` and a `success: false` payload.

In full (non-laconic) mode, a `reportMeta` block is included in the `meta` object alongside the `traceId`. This snapshot lists the report's fields, parameters, and resources at the time of the failure, useful for diagnosing mismatches between the request payload and what the report actually expects:


```
{
      "No data delivered (or fetched via SQL using parameters) while data deliverance i
mandatory      for      reports      with      detail      bands      or      SQL      queries."
  ],
                                        {     "parameterName":     "P_RESOURCE_LOGO",     "fileN
      ],
                                  {    "parameterName":    "P_ARTICLE_NUMBER",    "dataType":
      ],
                                          {     "fieldName":     "articleNumber",     "dataTyp
                                          {     "fieldName":     "description",
                                        {     "fieldName":     "moq",
```

```
                                        {     "fieldName":     "deliveryTime",          "dat
                                        {     "fieldName":     "supplier",
                                          {     "fieldName":     "barcode",
      ]
    }
  },

}
```

If a `ReportHistoryRecord` was requested (`createHistoryRecord: true`), it is still created even when the render fails, the error is recorded in the history entry, which makes it possible to review failed renders from the frontend alongside successful ones.

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.

> ℹ **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.

Scan2Print: Printing labels by scanning

Scan2Print is a dedicated, mobile-optimized page for printing labels and documents straight from a barcode scan. It is built for handheld terminals (MDE) and smartphones, but works just as well on a desktop workstation with a USB scanner. Configure the station once, then every scan becomes a print job, no forms, no clicks, no PDF preview in between.

Typical use cases are inbound goods labels, article and batch labels, inspection tags or pallet labels, anywhere a worker scans a value and needs a printed label a second later.

Screenshot placeholder: `scan2print.index-mobile.png`
Scan2Print page on a handheld (approx. 360 px width): top bar with configuration summary and gear button, two scan fields, Print button.

How It Works

1. Open **Scan2Print** from the main navigation.
2. On the first visit, a configuration dialog opens. Select the report, the printer, the number of copies and fill in the fixed values for this station.
3. The page now shows one input field per scan parameter of the report, stacked on top of each other.
4. Scan the values. Every scan ends with **Enter**, which moves the cursor to the next field. The Enter on the last field triggers the render.
5. VeloxFactory renders the report, creates a print task and returns to the empty scan form. The cursor is back in the first field, ready for the next scan.

Under the hood, Scan2Print uses the same rendering pipeline as the Generate PDF page and the API. Every Scan2Print submit is a regular render request with a print task attached, so everything you know about print tasks, history records and SQL data adapters applies here as well.

Permissions

Scan2Print has its own permission, found in the user settings under the section **Scan2Print**:

Permission	Purpose
`scan2print:use`	Shows the Scan2Print navigation item and grants access to the page. Administrators (`global:admin`) have access automatically.

Screenshot placeholder: `scan2print.user-permission.png`
User edit page, section *Scan2Print* with the *Use* checkbox.

`scan2print:use` is all a scan user needs. Every Scan2Print submit runs through the regular rendering pipeline, which checks its own permissions for rendering, print tasks and history records. The Scan2Print permission therefore implicitly includes these permissions, there is no need to grant them separately:

Implicit permission	Required for
Report Configs: **Read**	Selecting a report and rendering it.
Report Print Tasks: **Read** and **Create**	Creating the print job after every scan and checking the print queue.
Report History Records: **Create** and **Update**	Storing the render in the report history, if enabled in the station configuration.

As a consequence, a Scan2Print user also sees the **Reports** and **Print Queue** items in the navigation and can open report configurations read-only, including their Generate PDF page.

> ℹ **Dedicated scan users land on Scan2Print directly.** Users who hold `scan2print:use` but may not create or edit report configurations are redirected to Scan2Print when they open the application root. Bookmarking the Scan2Print page on the device works as well, after a login VeloxFactory returns to it automatically.

Preparing a Report for Scan2Print

Scan2Print does not need a special report type. It works with any report configuration and decides by **parameter name prefix** which values are scanned and which are fixed:

Prefix	Role	Filled in
`P_SCAN_`	Scan field. Appears as an input on the scan page. Always mandatory.	On every scan
`P_STATIC_`	Fixed value for the station, e.g. client name, storage location or inspector.	Once, in the configuration dialog
`P_RESOURCE_`	Image resource, handled automatically as usual.	Never
any other	Regular parameter. Not filled by Scan2Print, but can be delivered by the SQL query (see below).	Never

The scan fields appear in the order in which the parameters are declared in the `.jrxml`. The field label is derived from the parameter name, `P_SCAN_ARTICLE_NUMBER` is shown as **Article Number**. Scan and static fields use the same typed inputs as the Generate PDF page, a `java.sql.Date` static parameter becomes a date picker, a `java.lang.Integer` scan field a numeric input.

A minimal parameter block looks like this:

```
<parameter                name="P_SCAN_ARTICLE_NUMBER"                class="java.lang.String">
                                                                                        <pro


</parameter>
<parameter                 name="P_SCAN_ARTICLE_BATCH"                class="java.lang.String">
                                                                                    <property


</parameter>
<parameter                name="P_STATIC_INBOUND_DATE"                class="java.lang.String">
                                                                                      <property


</parameter>
```

Which reports can be selected

Only reports that Scan2Print can actually fill are offered in the configuration dialog. A report is selectable if **all** of the following apply:

- It has at least one `P_SCAN_` parameter.
- None of its `P_SCAN_` parameters is of type `java.lang.Boolean`, a toggle cannot be scanned.
- It has **no detail band**, or it has an **SQL data adapter**. Scan2Print never sends report lines, so detail data can only come from a query.
- It has no **required** parameter outside `P_SCAN_` and `P_STATIC_`, because Scan2Print has no way to fill it.

If your report does not show up in the dropdown, check it against this list first.

Enriching the label with SQL

The real strength of Scan2Print comes with an SQL data adapter. Scan only the key, for example an article number, and let the query fetch everything else. Scan values are available in the query like any other parameter, and every column aliased with a parameter name is transferred into that parameter:

```
select

                                    ,                   b.barcode
                                   ,                    a.supplier
from                                          article                                       a
inner       join       articlebarcode       b       on       a.id       =       b.articleId
where                a.articleNumber                  =                :P_SCAN_ARTICLE_NUMBER
```

Parameters that are filled by the query must be marked as **not required** in the `.jrxml`. VeloxFactory validates required parameters before the query runs, so a required parameter that only the query delivers would always fail.

> ℹ **A query must return at least one row.** If a report has an SQL data adapter and the query finds nothing, for example because an unknown article number was scanned, the render fails with "*No data delivered (or fetched via SQL using parameters)...*". No print task is created, so no half-empty label is printed. If the query returns several rows, the values of the last row are used for the parameters.

Configuring the Scan Station

The configuration dialog opens automatically when no configuration exists yet. Afterwards, it can be reopened at any time with the **gear** button in the top bar of the page, which also shows a compact summary of the active configuration (report, printer, copies, broadcast ID and static values).

Option	Default	Description
Report	-	Searchable list of all Scan2Print capable reports. Required.

Option	Default	Description
Printer Name	-	The target printer for every print task of this station. Required.
Copies	1	Number of copies per scan, between 1 and 100.
Broadcast ID	-	Optional WebSocket channel ID for real-time notification of the print service. Leave empty to rely on polling.
Create History Record	**On**	Stores every scan render in the report history, including request, response and PDF. Can be switched off for high-volume stations.
Static Parameters	-	One typed input per `P_STATIC_` parameter of the selected report. They are loaded as soon as a report is selected and replaced when you switch to another report. All of them are required, except boolean toggles.

Screenshot placeholder: `scan2print.config-modal.png`
Configuration dialog with selected report, printer, copies, broadcast ID, history toggle and loaded static parameters.

Click **Save** to store the configuration. If something is missing or invalid, VeloxFactory shows an error message and reopens the dialog with your input still in place.

To reset the station, open the dialog and click **Clear configuration**. The scan form disappears and the dialog asks for a new configuration.

> **i Static values are fixed.** A static date stays exactly as entered, it does not move on to the next day. If a label should always show the current date, use a default expression in the `.jrxml` instead of a static parameter.

Scanning and Printing

With a valid configuration, the page shows all scan fields of the report, large and full width, with the cursor already in the first field.

- **Enter** in a filled field moves the cursor to the next field and selects its content.

- **Enter** in the last field submits the form and starts the render.
- **Enter** in an empty field does nothing, the cursor stays where it is. All scan fields are mandatory.
- The **Print** button below the fields submits the form as well, which is handy for manual input on a desktop.

Screenshot placeholder: `scan2print.scan-form.png`
Scan form on desktop with a configured report, cursor in the first scan field.

After submitting, the form is locked until the page has reloaded. Scanners that send their data in quick bursts or add an extra Enter therefore never create duplicate print jobs. Reloading the page after a print does not print again either.

The Result

Success

A short green notification confirms the print job for two seconds, including report name, printer and number of copies. The scan fields are empty again and the cursor is back in the first field. The print task is now in the print queue and will be picked up by the print service.

Screenshot placeholder: `scan2print.result-success.png`
Green notification after a successful scan, empty scan fields.

Errors

If the render fails, a red block appears in the page body. It stays visible until the next scan and contains:

- all error messages returned by VeloxFactory, e.g. an unknown article number or a missing value,
- the values that were scanned, so the worker knows which item failed,
- a link to the print queue.

The scan fields are cleared and the cursor is back in the first field, so the worker can simply scan again.

Screenshot placeholder: `scan2print.result-error.png`
Red error block with error message (e.g. unknown article number), scanned values and link to the print queue.

> **i Check the print queue before rescanning after a WebSocket error.** If the broadcast to the print service fails, for example because Reverb is not running, the print task has already been created and is still printed by polling. Scan2Print shows an error in this case, a rescan would print the label twice.

Where the Configuration Is Stored

The station configuration is stored in the **user session** on the server. This keeps each device independent, two handhelds logged in with different users can serve different reports and printers at the same time.

- The configuration survives page reloads and navigation within VeloxFactory.
- It is removed on logout and when the session expires after the configured idle time (`SESSION_LIFETIME`). The worker then logs in again and reconfigures the station. After a session timeout, a pending scan cannot be submitted anymore and the page has to be reloaded.
- Static values are stored by parameter name. If the report is deleted, loses its Scan2Print capability, or a new `.jrxml` with additional required static parameters is uploaded, the configuration is reset automatically on the next page load, a warning is shown and the configuration dialog opens.

Scanner Requirements and Tips

- Scan2Print expects a scanner in **keyboard mode** (keyboard wedge), as used by nearly all handheld terminals and USB scanners.
- Configure the scanner to send **Enter** as suffix. A **Tab** suffix only moves the cursor and never submits the form.
- Camera scanning is not part of Scan2Print. Smartphones need a hardware scanner or a scanner app that types into the focused field.
- On smartphones, the on-screen keyboard opens when a field is focused. This is intended, it keeps manual input possible.

- For barcodes on the printed label, prefer **Code128** when the label should reproduce the scanned value exactly. EAN-13 components pad shorter values such as EAN-8 codes with zeros and recalculate the check digit, so the printed barcode would differ from the scanned one.