Managing reports in VeloxFactory

A report in VeloxFactory is more than a file. It is a fully managed configuration - with its own parameters, data fields, image resources, SQL query, data connection, preview images, rendering history, and print records. This page walks through the complete lifecycle of a report configuration, from upload to print.

Screenshot placeholder: `report-config.index.png`
Report Config overview, the landing page after login, with context badges and thumbnails.

The ReportConfig - Central Master Data

The `ReportConfig` is the core entity in VeloxFactory. Every report you manage is a `ReportConfig` record, and everything else in the system either belongs to it or references it. A single `ReportConfig` brings together:

- The `.jrxml` **template file** stored on disk
- Its **parameters** - input values passed at render time
- Its **fields** - data columns that populate the detail band
- Its **resources** - graphic assets (images, logos) embedded in the template
- Its **SQL query** - defined and managed directly in VeloxFactory
- Its **data connection** - the live database to query (optional)
- Its **context** - an organisational label for grouping
- Its **preview and thumbnail images** - generated from example data

Nothing renders without a `ReportConfig`. Nothing prints without one either. It is the starting point for every operation in VeloxFactory.

The Report Configuration Lifecycle



Each step is described in detail below.

Uploading a Report

When you upload a `.jrxml` file, VeloxFactory immediately analyses it and builds the initial configuration automatically. The following is extracted from the file:

- **Report name** - from the `name` attribute on `<jasperReport>`. Must be unique.
- **Page dimensions** - width and height, converted from Jasper pixels to millimetres.
- **Detail band presence** - whether the report has a repeating data section.
- **Parameters** - all non-resource parameters, including their data types and any `exampleValue` / `required` custom properties set in the `.jrxml`.
- **Fields** - all data fields, including their data types and `exampleValue` custom properties.
- **Resources** - all parameters following the `P_RESOURCE_` naming convention (image assets).

The result is a fully structured `ReportConfig` record with all its child records in place, but not yet complete. Resource files still need to be uploaded, and the SQL query still needs to be written if a live data connection is used.

> ⚠ **Report name and file name must be unique.** VeloxFactory will reject an upload if a `ReportConfig` with the same report name or the same file name already exists.

The file name may contain letters, digits, dot, hyphen and underscore, must end in `.jrxml` and is limited to 120 characters. Directory parts are not accepted.

Screenshot placeholder: `report-config.create.png`
Report Config create form with the JRXML upload, context and data adapter.

Completing the Configuration

After upload, the report configuration is ready but not yet fully operational. The following steps complete it:

Upload Resource Files

If the report contains image resources (detected as `P_RESOURCE_` parameters), each one requires an actual file to be uploaded. VeloxFactory cannot render the report until all resource files are in place.

Alternatively, a resource can be linked to a `CommonReportResource` - a shared asset reused across multiple reports, such as a company logo. Linking is permanent: the resource's own file is deleted and the common file is used in its place.

A Common Report Resource serves a second purpose as well: a PNG or JPEG uploaded there can be placed into a mail template as an inline graphic, referenced by its name. The same logo therefore covers the report and the mail it is sent with. See Mail Templates.

Resource files are images: `png`, `jpg`, `jpeg`, `gif` or `webp`. VeloxFactory checks the actual content, not the extension, and refuses a file whose bytes are not one of those formats.

The file name is normalised before it is stored. Any directory part is dropped, inner dots become hyphens (`logo.v2.png` is stored as `logo-v2.png`), everything outside letters, digits, hyphen and underscore becomes a hyphen, and the name part is cut at 80 characters. The stored name is what the report refers to, so a renamed file shows up under its normalised name in the resource list.

> ℹ **The linked/uploaded file is the default, not the only option.** A single render request can override a resource's image for that call alone via `resourceOverrides` - a local path, a remote URL, or a Base64 upload - without touching this configuration. The file uploaded here is still required as the fallback. See Rendering with our powerful API.

Screenshot placeholder: `report-config.edit.png`
Report Config edit view with parameters, fields and resources.

Review Parameters and Fields

VeloxFactory picks up `exampleValue` and `required` custom properties from the `.jrxml` automatically on upload. If these were not set in Jaspersoft Studio, or if you need to adjust them, you can do so directly in the configuration.

Every parameter and field should have an example value set before generating a preview.

Write the SQL Query and Assign a Connection

If the report fetches live data from a database, assign a `ReportConnectionConfig` and write the SQL query in the **Query** field. The query is stored in VeloxFactory, not in the `.jrxml`.

Parameters are available as named bindings in the query (`:PARAMETER_NAME`). See Creating reports in Jaspersoft Studio for details on how parameter binding works.

A connection is not required if the report has no detail band, or if data will be delivered in the render request itself.

Generating a Preview

Once all resource files are uploaded and all parameters and fields have example values, you can generate a preview. VeloxFactory renders the report using the stored example values, no live data needed, and stores the result as a base64-encoded PDF and a thumbnail image on the `ReportConfig`.

The preview is used in the report list as a visual card and as a quick sanity check that the template renders correctly. It is also what the `useExampleValues` flag triggers during a render request, useful

for testing without providing real data.

> ℹ **Preview generation will fail if any resource file is missing or any example value is not set.** VeloxFactory checks all three conditions - resources, parameter example values, and field example values - before attempting to render.

Screenshot placeholder: `report-config.edit.preview.png`
Report Config edit view with the generated preview and thumbnail.

Report History Records

Every render request can optionally create a `ReportHistoryRecord` - a full log entry of what was requested and what was returned. This is controlled by the `createHistoryRecord` flag in the render request body and is **off by default**.

When enabled, the history record captures:

- The exact request payload sent to the render endpoint
- The full API response
- The rendered PDF (Base64-encoded)
- A thumbnail of the first page (generated asynchronously in the background)
- The rendering status (`ok`, `render_fail`, `error`, `unknown`)
- The trace ID for cross-referencing with logs

History records are valuable for traceability, you can see exactly what was rendered, when, with what data, and what the result was. From a history record, you can also dispatch a reprint or send the document by mail directly.

> ℹ **Downloads follow the read permission.** Downloading a report template, a resource file, a rendered PDF or an xlsx export requires the read permission for that resource. Being signed in is not enough.

When to Skip History Records

History records are entirely optional. There are two good reasons to leave them off:

Performance and storage. Storing the full PDF, request payload, and response for every render adds up. For high-frequency rendering where traceability is not needed, skipping history records keeps the database lean.

Data sensitivity. A history record stores the complete render payload, including all parameters and data passed to the report. If that data is sensitive (personal data, financial figures, medical information), you may not want it persisted on the server at all. Omitting `createHistoryRecord` from the render request ensures nothing is logged.

Retention

History records are automatically purged after a configurable number of days. The retention period is set via the `PURGE_HISTORY_DAYS` environment variable (default: 30 days). Purging runs automatically as a background job, no manual intervention required.

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

Report Mail Tasks

The second way out of VeloxFactory is a mail. A render request can carry a `mailing` segment, and a document that was rendered earlier can be mailed from its history record at any time. Either way a `ReportMailTask` is created, logging recipients, subject, body and attachments, exactly as a print task logs printer and copies.

What a mailing needs is set up once under *Configuration*: a **Mailer** holds the SMTP account, a **Mail Template** holds subject and body with placeholders. The PDF is attached, and on request an xlsx export of the same rows the report was rendered with. Sending is synchronous by default, or handed to the queue.

> **i A mail task blocks its history record from being deleted, just like a print task does.** Mail tasks have their own retention setting, `PURGE_MAILTASKS_DAYS`, and are purged before the history records they point at. Full detail on Report Mailing.

Report Print Tasks

A `ReportPrintTask` sends a rendered PDF to a physical printer. Print tasks are created as part of a render request, you render and dispatch to a printer in a single call, by setting `createPrintTask: true` and providing a `printerName`.

Print tasks are always linked to a `ReportHistoryRecord`. This means creating a print task also creates a history record (regardless of whether `createHistoryRecord` is explicitly set), so the printed document is always traceable.

How Printing Works

VeloxFactory does not communicate with printers directly. Instead, it creates a `ReportPrintTask` record and notifies a separate print service - a lightweight C# application running on or near the target machine - which picks up the task and executes the print job.

There are two modes of delivery:

WebSocket (push). If a `broadcastId` is included in the render request, VeloxFactory broadcasts a `ReportPrintTaskCreated` event via WebSocket (Laravel Reverb) the moment the task is created. The print service subscribes to that channel and reacts immediately. This is the recommended mode for real-time printing, the task reaches the printer within milliseconds of the render completing.

Polling (pull). Without a `broadcastId`, no broadcast is sent. The print service must poll the API for new tasks in `pending` status. This works fine for less time-sensitive workflows.

Print Task Status

Status	Meaning
pending	Created, waiting for the print service to pick it up
printed	Print job executed and confirmed by the print service
error	Print service reported a failure
unknown	Status could not be determined

The print service reports status back to VeloxFactory via the API after executing the job. The `error_message` field on the task record contains the failure detail if printing did not succeed.

Copies

The `numberOfCopies` field is passed to the print service as the requested number of printed copies. It defaults to `1` if not specified. VeloxFactory always renders the PDF exactly once, the print service

is responsible for duplicating the output on the printer side.

> ℹ **Print tasks also have configurable retention.** They are automatically purged after `PURGE_PRINTTASKS_DAYS` days (default: 30). Like history record purging, this runs in the background without any manual action.

Screenshot placeholder: `report-print-task.index.png`
Report Print Task overview with status badges, printer and the reset action.

Deleting a Report Configuration

A `ReportConfig` can only be deleted when no `ReportHistoryRecord` or `ReportPrintTask` references it. VeloxFactory will reject a deletion request while any such records exist.

When a `ReportConfig` is deleted, the following is removed along with it: the `.jrxml` file from disk, all resource files, and all parameter and field records. The deletion is atomic, if any step fails, the entire operation is rolled back.

> ⚠ **To delete a ReportConfig that has history records or print tasks, those records must be removed first.** Once the retention period has passed and the automatic purge has run, or once the records are manually deleted, the ReportConfig can be removed.

Revision #15
Created 2026-05-10 10:39:30 UTC by Benjamin Fischer
Updated 2026-09-22 20:53:21 UTC by Benjamin Fischer