Mail your renderings

Sending rendered reports by mail: the mailing segment, mailers, mail templates and the mail queue.

- Report Mailing: sending renderings by mail
- Mailers: master data for your SMTP accounts
- Mail Templates: subject, body and whitelabeling

Report Mailing: sending renderings by mail

A rendered report usually has somewhere to go: back to the caller as Base64 or as a URL, straight into a print task, or into the mailbox of whoever needs the document. **Report Mailing** is that last destination. A render request carries an optional `mailing` segment, and once the render succeeded VeloxFactory sends an HTML mail through a configured mailer, using a configured template, with the PDF attached and, if you want it, an xlsx export of the same data.

Mailing is an addition to a render, never a condition of it. A request without a `mailing` segment is a plain render, and a mail that fails never turns a successful render into an error.

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

The three parts

Mailing is built from three things you configure once and one thing you send per render.

Part	What it holds
Mailer	The SMTP account: host, port, credentials, sender address, and the dispatch rate limits your provider allows. Master data under Configuration.
Mail Template	Subject and body, written in a WYSIWYG editor, with placeholders and optional whitelabeling. Master data under Configuration.
Mailing segment	Per request: which mailer, which template, who receives it, what is attached, and whether it goes out immediately or through the queue.
Report Mail Task	The record of one send attempt: recipients, the rendered subject and body, the attachments, the status and, if something went wrong, the error.

Mailing from a render request

The `mailing` segment is optional and sits next to the fields you already know:

```
{
                                                    "parameters":              {
                                                                                  }
}
```

`mailer` and `mailTemplate` accept either the numeric ID or the unique name of the record, so a request stays readable. Only active records are resolved.

Field	Required	Description
mailer	yes	ID or unique name of an active Mailer.
mailTemplate	yes	ID or unique name of an active Mail Template.
to, cc, bcc	to: yes	Address lists. Literal addresses are validated before the render is spent, addresses containing a placeholder are resolved afterwards.

Field	Required	Description
contactName	no	Free text for a personal greeting, available in the template as `[contactName]`. May itself be a placeholder.
includePdf	no	Default `true`. Set it to `false` for a notification mail that only carries the xlsx, or nothing at all.
pdfFileName	no	Base name the PDF carries as an attachment of the mail, without extension. The stored file keeps its own name, nothing is copied.
includeExcel	no	Default `false`. Builds an xlsx from the rows that were actually rendered.
includeParameters	no	Prints the parameters as a block above the data table in the xlsx.
excelFileName	no	Base name the xlsx carries as an attachment, same rules as `pdfFileName`.
sendAsync	no	Default `false`. `true` hands the mail to the queue and answers immediately.

> ℹ **A failed mail never fails the render.** Whatever goes wrong with the mailing, the render response stays a success and the detail lands in `meta.mailing` and on the mail task. Your integration keeps its PDF.

The one combination that is refused up front is `outputType: "preview"` together with `mailing`: a preview render deletes its own file immediately, so there would be nothing to attach.

The Excel attachment

`includeExcel` turns the data of the render into a real xlsx, not a CSV with a different name. The rows come from what was actually rendered, which matters for reports fed by an SQL adapter: there the rows only exist in the render result, never in your request.

The sheet has a header row in the report's accent colour, frozen below the header, with an auto filter across the table. Columns are typed from the report's field definitions, so an integer is a number, a date is a real Excel date and a decimal keeps its format. For a dynamic array without field definitions the type is inferred from the values. Rows with differing keys are merged into one header, in first-seen order, and missing values stay empty.

With `includeParameters` the parameters are printed above the table, so the recipient can see what the report was asked for without opening the mail again.

If the render produced a history record, the xlsx is stored on it and stays downloadable from the history record page. It is cleaned up by the same retention as the PDF.

Placeholders in recipients and attachment names

Subject and body are placeholder-driven, and so are the **recipients**, the **contact name** and the **attachment names**. They all accept the same catalog, which is what makes mailing useful for data that only exists after the render.

Every route that takes recipients accepts them: the `mailing` segment of a render, `POST /report-history-record/{id}/mail`, `POST /report-mail-task` and the repeat endpoint. The failure recipients of a schedule are the exception, they are plain addresses, because a run that failed has no render to resolve against.

```
"to":                                  ["[data.first.customerEmail]"],
"contactName":                           "[data.first.customerName]",
"pdfFileName":                 "Lieferschein_[parameters.P_ORDER_NO]_[now.date]"
```

For a report with an SQL adapter, `[data.first.<column>]` addresses the rows the query fetched. The recipient list is therefore a result of the render, not an input to it.

A few rules are worth knowing:

Situation	Behaviour
A placeholder holds several addresses	Comma or semicolon separated values are split, trimmed and de-duplicated. One column can address a whole distribution list.
A `cc` or `bcc` placeholder resolves to nothing	The address is left out and the mail goes anyway. A render reports it in `meta.mailing.unresolvedPlaceholders`, `POST /report-mail-task` as an *Unresolved placeholder* note in `meta.messages`.
No valid `to` address is left, or one resolves to something that is not an address	In a render no mail is sent, the reason is in `meta.mailing.errors` and the render itself is unaffected. On `POST /report-mail-task` the call answers 422 and no task is created.

Situation	Behaviour
What the mail task stores	The resolved address list, not the placeholder. `[recipients.toCount]` and the other counters in subject and body therefore count real recipients.
Column names	Matched exactly as the adapter returns them. `[data.first.EMAIL]` is not `[data.first.email]`.
Attachment names	Resolved, then made file-system safe with case and underscores preserved. Umlauts are transliterated, invalid characters collapse to a single hyphen, the extension is forced.

> ℹ **Placeholder values are inserted as text.** A resolved value that happens to contain markup, for example a customer name with a `<b>` in it, appears literally in the mail instead of changing its formatting. The markup you set in the editor is unaffected.

> ℹ **The name is a label, not a file.** `pdfFileName` and `excelFileName` only change what the attachment is called in the mail. On disk the PDF and the xlsx are always stored under the trace id, they hang on the history record, stay downloadable from it and are deleted with it. Two mails may carry the same attachment name without anything being overwritten.

Without a requested name the attachment is called like the stored file, which is the trace id plus its extension.

Immediately or through the queue

By default the mail is sent while the request is still open. The response then already tells you whether it went out. That is the right mode for a single document and a fast relay.

`sendAsync: true` creates the mail task, hands it to the queue and answers immediately. The mail is sent by a Horizon worker, the task moves from *Pending* to *Sent*, and a failure is retried with backoff before it ends as *Error*. Use it for bulk runs and for slow or unreliable SMTP servers, where waiting for the transport would block the render.

> ⚠ **Asynchronous sending needs a running queue.** Without `php artisan horizon` the mail task stays *Pending* forever. It is visible in the Mail Queue, so nothing is lost, but nothing is sent either.

There is a third case you do not ask for: if the mailer's rate limit is exhausted, a synchronous send is converted into a delayed queued send instead of failing. The mail is not lost, the response says

so, and the task shows when it will go out. See the Mailers page for the windows.

Mailing from the frontend

The same settings are available in four places in the UI, built from one shared block of fields, so they look and behave identically everywhere.

Place	How it works
Generate PDF	A **Send results via Mail** toggle in the Output Settings. Configure it once, and every render from that page mails its result.
Scan2Print	The same toggle in the station configuration. Every scan then prints and mails.
Report History Record	A **Send Mail** button that mails a document which was rendered earlier, with a freely chosen mailer, template and recipients.
Job Scheduler	The same toggle on a render job. Every run of the schedule mails its result.

In the configuration modals the mailing settings live in the session alongside the rest of the output settings, and the values are kept even while the toggle is off, so switching mailing off for a moment loses nothing. The file name fields appear only while their attachment switch is on.

After a render the notification says what happened to the mail. A mail that did not go out as configured turns the success notification into a warning instead of hiding the problem.

The block is only rendered at all when the current user may create mail tasks and at least one active mailer and one active template exist. Nobody is offered a button that cannot work.

Screenshot placeholder: `mailing.generate-pdf-settings.png`
Output Settings of the Generate PDF page with the mailing block expanded.

On a schedule

A mail does not need a caller. A render job in the **Job Scheduler** carries the same mailing settings as a render request, so a schedule puts the shift report in a mailbox at six every morning without anybody asking for it.

Two things work differently from an interactive send. The schedule's owner and API token decide what may go out: the token needs `report-mail-task:create`, and a revoked token or a deactivated owner parks the schedule rather than mailing. And placeholders are resolved at the moment the run fires, so `[now.date]` in a subject line or an attachment name carries the date of that run, not the date the schedule was saved.

Every run is logged with its trace id. A mail task that ends in *Error*, or anything reported in `meta.mailing`, marks the run as a warning, so a schedule whose mail is not arriving is visible in the run log without opening the Mail Queue.

> **i Two different mails.** The mail configured here is the document itself. A schedule that could not be executed at all sends the Job Scheduler's own failure mail to its own recipients. See *Schedule your jobs*.

The Mail Queue

Every send attempt, synchronous or queued, is recorded as a **Report Mail Task**. The overview lives in the main menu under **Mail Queue** and is a regular VeloxFactory lookup with the usual filters and the usual session-persisted state.

A mail task keeps what was actually sent, not just what was requested: the resolved recipients, the rendered subject, the rendered body and the names of the attachments. The detail page shows the mail exactly as the recipient received it, rendered in an iframe from the stored HTML, whitelabeling included.

Status	Meaning
Pending	Queued, or waiting for a free rate limit slot. The task shows when it is due.
Sent	The transport accepted the message, with the timestamp. Delivery itself is not tracked.
Error	The send failed after its retries. The transport's own message is on the task.

Repeat resends a task. It opens a dialog in which mailer and recipients can be changed before sending, so a mail that went to the wrong address is corrected rather than cloned. Subject and body are re-rendered from the current template when the history record still exists, otherwise the

stored snapshot is reused. The repeated task gets its own trace id, derived from the original.

The same table also appears on the Report History Record page, so the mails belonging to one document are visible where the document is.

Screenshot placeholder: `mailing.mail-queue.png`
Mail Queue overview with status badges, recipients and the repeat action.

Retention

Mail tasks are cleaned up by the **Purge Job**, the consolidated cleanup schedule of the Job Scheduler. It removes print tasks and mail tasks before the history records they reference, so nothing is deleted out from under a reference. How long mail tasks are kept is a retention on that schedule, editable in the frontend or through the API like everything else; an empty value switches the step off entirely.

The same work is available as a standalone command:

```
php artisan report-mail-tasks:purge --days=30
```

Attachments are not owned by the mail task. The PDF and the xlsx belong to the history record and are removed by its retention step, or, if no history record was created, by the orphaned-file step.

The API

Besides the `mailing` segment on `POST /report-config/{id}/render`, mailing has endpoints of its own.

Endpoint	Description
`POST /report-history-record/{id}/mail`	Mail a document that was rendered earlier. Takes the same fields as the mailing segment, including placeholders in recipients and attachment names. Builds the xlsx on the fly if the record does not have one yet.
`GET /report-mail-task`	List mail tasks, filterable by status.

Endpoint	Description
`GET /report-mail-task/{id}`	A single mail task, with its relations and its audit segment on request.
`POST /report-mail-task`	Create and dispatch a mail task directly. `to`, `cc` and `bcc` take literal addresses or placeholders, resolved against the render of the referenced history record.
`POST /report-mail-task/{id}/repeat`	Resend a task. Optional `mailer`, `to`, `cc` and `bcc` override the original values, anything left out is reused. The overriding recipients accept placeholders too, resolved against the history record behind the task; if the task has none, only the generic placeholders such as `[user.email]` resolve.
`DELETE /report-mail-task/{id}`	Delete a mail task. The attachments belong to the history record and are not touched.

A render that mailed something returns the mail task in its response, and anything noteworthy about the mailing in `meta.mailing`:

Key	Meaning
`errors`	The mail was not sent, with the reason.
`skipped`	The render failed, so there was nothing to send.
`excelError`	The xlsx could not be built. The mail goes out with the PDF only.
`unresolvedPlaceholders`	Placeholders in recipients, contact name or attachment names that resolved to nothing.

AI assistants

The MCP server exposes mailing the same way it exposes printing: the `mailing` segment is part of the render tool, and mail tasks, mailers and templates have their own tools. Because mailer and template are resolved by name, an assistant can pass what a person said. *Render the delivery note for order 4711 and mail it to dispatch* is one call.

Permissions

Mailing has three permission scopes, each with the usual `read`, `create`, `update`, `delete` and `full`.

Scope	Grants
`mailer:*`	The Mailers master data.
`mail-template:*`	The Mail Templates master data.
`report-mail-task:*`	Seeing the Mail Queue, and `create` for actually sending mail, from the API as well as from the frontend.

`report-mail-task:create` is the one that matters for sending. It is checked at the start of a render, before anything is rendered, so a user without it gets a clear refusal instead of a wasted render.

Mail graphics are Common Report Resources, so maintaining them uses `common-report-resource:*`. That permission also covers the logos printed on labels, which is the price for uploading a logo once rather than twice.

Mailers: master data for your SMTP accounts

A mail needs an account to go out through. **Mailers** are that account as master data: one record per SMTP mailbox or relay, with its host, its credentials, the address it sends from, and the limits your provider actually enforces. Every mail VeloxFactory sends goes through one of these records.

The `MAIL_*` keys in the environment file are Laravel's own fallback mailer and are deliberately **not** used by report mailing. Mail configuration belongs in the application, where it can be changed, tested and audited without a deployment.

Screenshot placeholder: `mailer.index.png`
Mailers overview under Configuration with transport badges, the approval status and the Active column.

Where to find it

Mailers live in the main menu under **Configuration → Mailers**. The overview is a regular VeloxFactory lookup: search over name, host, sender address and description, filters for transport, active state, tested state, creator, updater and creation date, sortable columns, and the filter state kept in your session.

New opens the create form, the pencil on a row opens an existing record. Deleting is done from the edit page.

The fields of a mailer

Field	Required	Description

Name	yes	Unique, three to 55 characters, for example `Office SMTP`. This is what a render request may use instead of the ID.
Transport	yes	`smtp` for a real server, `sendmail` for the local binary, `log` to write the mail into the log instead of sending it. `log` is what makes a demo or a staging system harmless.
Host, Port	for smtp	The server and its port, for example `smtp.office365.com` and `587`.
Scheme	no	`smtp` for STARTTLS on port 587, `smtps` for implicit TLS on port 465. Left empty the transport decides by port, which is right for most servers.
Username, Password	no	Stored encrypted. The password is never returned by the API, and the edit form shows it empty: leave it empty to keep the stored one.
From Address	yes	The sender of every mail sent through this mailer. Many providers require it to match the authenticated mailbox.
From Name	no	The display name next to the address, for example `Shipping Department`.
Reply To	no	Where replies should go, when that is not the sender. Useful for a no-reply sender with a real support mailbox behind it.
Timeout	no	Seconds before the transport gives up. Relevant for synchronous sending, where the render request waits.
Local Domain	no	The EHLO name. Left empty the host part of the application URL is used. Some strict relays insist on a specific value.
Limit per minute / hour / day	no	The dispatch rate limits, see below. The hourly limit defaults to 500.
Description	no	Free note, for example which department the mailbox belongs to.
Active	-	On by default. An inactive mailer stays in the master data and keeps its history, but it can no longer be selected and no longer resolves in a render request.

Screenshot placeholder: `mailer.edit.png`
Mailer edit form with connection fields, the three rate limit fields and the remaining-slots badges.

Testing a mailer

A new or changed mailer starts as **Unapproved**. The edit page has a **Send Test Mail** button that sends a short message through exactly the configuration you just saved, to your own address by default. On success the mailer flips to **Approved**, on failure the transport's own error message is shown, which is usually enough to see whether it was the port, the credentials or the TLS mode.

This mirrors how report connections are verified, and it works the same way: the status is information, not a gate. A mailer that has never been tested can still be used, it is simply not marked as proven.

> ℹ **Changing a connection field resets the approval.** Transport, host, port, scheme, credentials and sender address all invalidate the tested state, so a green badge always refers to the configuration that is actually stored.

A test mail consumes a rate limit slot like any other mail, so the counter stays honest. If the limit is exhausted, the test is refused with the waiting time instead of being sent.

Rate limits

Providers cap outbound mail, and they all cap differently. Exceeding the cap does not produce a polite error, it produces a blocked account, so the limits belong on the mailer, next to the credentials they protect.

Each mailer has three independent windows, each of them optional:

Window	Typical for
per minute	Relays that cap bursts, for example Office 365 at roughly 30 messages per minute.

Window	Typical for
per hour	Shared hosting mailboxes, typically 200 to 500 per hour. This is the one that defaults to 500.
per day	Mailbox providers, for example Gmail at 500 per day.

An empty field means the window is not enforced. All three empty means the mailer sends unthrottled. Check your provider's documentation and configure slightly below its cap, because the windows here are fixed rather than sliding.

A slot is spent per **message**, not per recipient: a mail to five people costs one slot, which is how providers count as well.

When a slot is not available, VeloxFactory does not drop the mail:

Path	What happens
Synchronous	The mail is converted into a delayed queued send. The response says so, the task stays *Pending* and shows when it is due.
Asynchronous	The queued job releases itself until the window opens. Waiting does not count against the retry budget, so throttling never turns into an error.

The edit page shows the remaining slots per configured window, so a run that is about to hit a cap is visible before it does.

> ⚠ **The queue fallback needs a running queue.** If a synchronous send is pushed into the queue because of the rate limit and no Horizon worker is running, the mail waits as *Pending* in the Mail Queue. Nothing is lost, but nothing is sent either.

Counters are per mailer and per installation. Two VeloxFactory instances sharing one SMTP account do not share a counter.

Name resolution and deletion

A render request may address a mailer by its numeric ID or by its unique name, and only **active** records resolve. An unknown or inactive name is refused with a clear message before anything is rendered, rather than silently falling back to some default mailer.

A mailer cannot be deleted while a mail task still references it, so the history of what was sent stays intact. To retire a mailbox, switch the record to inactive: it disappears from every picker and

stops resolving, while everything that was sent through it keeps its record.

The API

Mailers are a regular API resource under `/api/v1/mailer`, with the same envelope, the same audit segment and the same error format as every other resource.

Endpoint	Description
`GET /mailer`	List mailers, optionally narrowed with `limit`.
`GET /mailer/{id}`	A single mailer, with `withAudit=true` including its audit segment.
`POST /mailer`	Create a mailer. `name`, `transport` and `fromAddress` are required, `host` and `port` additionally for SMTP.
`POST /mailer/{id}/test`	Send a test mail, optionally to a given `to` address, and update the approval status.
`PATCH /mailer/{id}`	Partial update. An omitted or empty password keeps the stored one.
`DELETE /mailer/{id}`	Delete a mailer, refused while mail tasks still reference it.

A mailer resource is returned as:

```
{
```

```
                        "rateLimitRemaining":       {       "minute":       28,       "h
}
```

The password is never part of the response.

AI assistants

The MCP server exposes the same operations as tools, including the test. An assistant can create a mailer from a provider's documented settings and verify it in the same conversation.

Permissions

Mailers use the standard permission scheme and have their own **Mailers** section in the user editor.

Permission	Grants
`mailer:read`	See the menu entry, the overview and a mailer's detail page. Also required to pick a mailer when sending.
`mailer:create`	Create mailers.
`mailer:update`	Change mailers, including the active state, and send test mails.
`mailer:delete`	Delete mailers.
`mailer:full`	All of the above, like `global:admin`.

Credentials are readable to anyone with `mailer:read` in the sense that the username is shown; the password is not, on any path. Treat `mailer:update` as an administrative permission.

Mail Templates: subject, body and whitelabeling

A mail needs something to say. **Mail Templates** are the subject and the body, written once and reused by every render that mails a document. They are edited in a WYSIWYG editor, so nobody has to write HTML, and everything variable is a placeholder that VeloxFactory fills in at send time: the trace id, a parameter of the request, the name of the report, a column of the data that was rendered.

A template decides the content. The surrounding mail, a plain card in your application's colours, is built by VeloxFactory, and a template can override parts of it when a customer needs their own look.

Screenshot placeholder: `mail-template.edit.png`
Mail Template edit page with the rich text editor, the toolbar and the placeholder and preview buttons.

Where to find it

Mail Templates live in the main menu under **Configuration → Mail Templates**. The overview is a regular VeloxFactory lookup with search over name, subject and description, a filter for the active state and the usual audit filters.

A template has a unique name, which is what a render request may use instead of the ID, a subject, a body and an active flag. An inactive template stays in the master data but is no longer offered and no longer resolves.

The editor

The body is written in a rich text editor built into VeloxFactory. No add-on, no external service, and nothing to learn: the toolbar does what a toolbar does.

Group	Tools
Text	Bold, italic, underline, headings, paragraph, bullet and numbered lists, text colour and remove colour, clear formatting.
Links	Insert and remove links. Only `http`, `https` and `mailto` are accepted, and every link is marked so it opens safely.
Tables	Insert a table with a chosen number of rows and columns and an optional header row, then insert or delete rows and columns from wherever the caret sits.
Images	Insert a graphic from the Common Report Resources, with an optional width.
Placeholders	Insert a placeholder at the caret, chosen from the catalog.

Pasting is deliberately plain text. Formatted content dragged in from Word or a browser is the single most common reason a mail looks broken in one client and fine in another, so the editor strips it and keeps the words.

Whatever the editor produces is cleaned on the server against an allowlist before it is stored. That is not a formality: it is what guarantees that a template cannot carry a script, a tracking pixel or an arbitrary external image, no matter what was pasted into it.

Placeholders

Anything in square brackets is replaced when the mail is built. Both the subject and the body support them.

> Hello [contactName], attached is your report [report.name] with the trace ID [traceId], rendered on [now.dateLocale].

The **Placeholders** button opens a dialog listing every token that is available, grouped, with a description and an example, and each one copyable with a click. Opened from a template that is tied to a report, the parameter and field tokens are expanded to the real names of that report.

Group	Examples

General	`[traceId]`, `[reportFileName]` (the attachment name), `[reportUrl]`, `[outputType]`, `[appName]`, `[contactName]`, `[broadcastId]`
Generic	`[now.date]`, `[now.dateLocale]`, `[now.dateTime]`, `[now.weekday]`, `[now.monthName]`, `[now.weekNumber]`, `[now.quarter]`
Report	`[report.name]`, `[report.description]`, `[report.context]`, `[report.connection]`
History Record	`[historyRecord.id]`, `[historyRecord.status]`, `[historyRecord.url]` as a direct link into VeloxFactory
User, API Token	`[user.name]`, `[user.email]`, `[token.name]` - who or what triggered the render
Printer	`[printer.name]`, `[printer.copies]` when a print task was created alongside
Parameters, Data	`[parameters.P_ORDER_NO]`, `[data.count]`, `[data.first.CUSTOMER]`, `[data.last.ARTICLE]`
Recipients	`[recipients.toCount]`, `[recipients.ccCount]`, `[recipients.bccCount]`
Environment	`[env.appEnv]`, `[env.hostname]`, `[env.timezone]` - to mark a mail from a test system as such
Images	`[image.logo_dark]` - one entry per usable Common Report Resource

> ℹ **An unknown placeholder never breaks a mail.** It resolves to an empty string and is reported back to the caller, so a typo shows up as a gap and a note, not as a failed send.

> ℹ **Placeholder values are inserted as text.** A resolved value that happens to contain markup, for example a customer name with a `<b>` in it, appears literally in the mail instead of changing its formatting. The markup you set in the editor is unaffected.

The same catalog is available beyond subject and body: recipients, contact name and the names the attachments carry in the mail accept placeholders too. See the Report Mailing page for how that behaves.

Graphics

Mail graphics are the **Common Report Resources** you already maintain, the same PNG and JPEG files the reports use. A logo is uploaded once and serves both a printed label and a mail.

Insert one with the image button, or type its placeholder, for example `[image.logo_dark]`. Both produce the same thing, and the editor shows the graphic while you write.

At send time the graphic is embedded **into** the mail as an inline attachment, not linked from a server. That matters for two reasons: the mail is self-contained, so it also works for a recipient outside your network, and no client shows a *load external images* bar or treats it as a tracking pixel.

> ℹ **Only graphics from the master data.** An image pasted in from a website is removed when the template is saved. If a graphic should be in a mail, upload it under Configuration → Common Report Resources and reference it by name.

A Common Report Resource that a template embeds cannot be deleted or renamed while the template uses it. The refusal names the template, so it is clear what would break.

Whitelabeling

By default every mail looks the same: a card in the colours of your VeloxFactory installation, read straight from the application's stylesheet, with a coloured header bar and a footer. Change a colour in the frontend and the mails follow, without touching a template.

When one customer or one process needs something else, a template can override parts of that frame. Each override has its own switch, and switching it off returns that part to the global default:

Override	Effect
Header colour	The colour of the bar at the top of the card, instead of the theme colour.
Body colour	The background of the content area, instead of white.
Logo	A different Common Report Resource in the header, instead of the configured logo.
Hide logo	No logo at all, for a plain header.
Footer text	A different footer line. Supports the full placeholder catalog, so it can carry a trace id, a contact or a legal note.

Screenshot placeholder: `mail-template.whitelabel.png`
The whitelabeling block with the per-field switches for header colour, body colour, logo and footer text.

Preview

The **Preview** button renders the template inside the real mail frame, with example values or against an existing history record, and shows it in a dialog. It is the same rendering path the actual mail uses and the same one the Mail Queue uses to show a sent mail, so the three cannot drift apart: what the preview shows is what is sent and what is archived.

Use it after every change. It is the fastest way to catch a placeholder that resolves to nothing, a table that lost its borders or a whitelabel colour that makes the text unreadable.

The API

Mail Templates are a regular API resource under `/api/v1/mail-template`.

Endpoint	Description
`GET /mail-template/placeholders`	The placeholder catalog, optionally expanded for a given `reportConfig`.
`GET /mail-template`	List templates.
`GET /mail-template/{id}`	A single template, with its audit segment on request.
`POST /mail-template`	Create a template. The body is sent Base64-encoded and is cleaned against the allowlist before it is stored.
`POST /mail-template/{id}/preview`	Render subject and body, with example values or against a history record, and report which placeholders stayed unresolved.
`PATCH /mail-template/{id}`	Partial update.
`DELETE /mail-template/{id}`	Delete a template, refused while mail tasks still reference it.

Unknown placeholders in a stored template are not an error. They are returned as messages, so an integration can warn without being blocked.

AI assistants

The MCP server exposes the same operations, including the placeholder catalog. An assistant can therefore write a template that uses the real parameter names of a specific report, and preview it before anyone sends it.

Permissions

Mail Templates use the standard permission scheme and have their own **Mail Templates** section in the user editor.

Permission	Grants
`mail-template:read`	See the menu entry, the overview and a template. Also required to pick a template when sending.
`mail-template:create`	Create templates.
`mail-template:update`	Change templates, including the whitelabeling and the active state.
`mail-template:delete`	Delete templates.
`mail-template:full`	All of the above, like `global:admin`.

Uploading the graphics a template uses is a separate permission, `common-report-resource:create` and `:update`, because those files are shared with the reports.