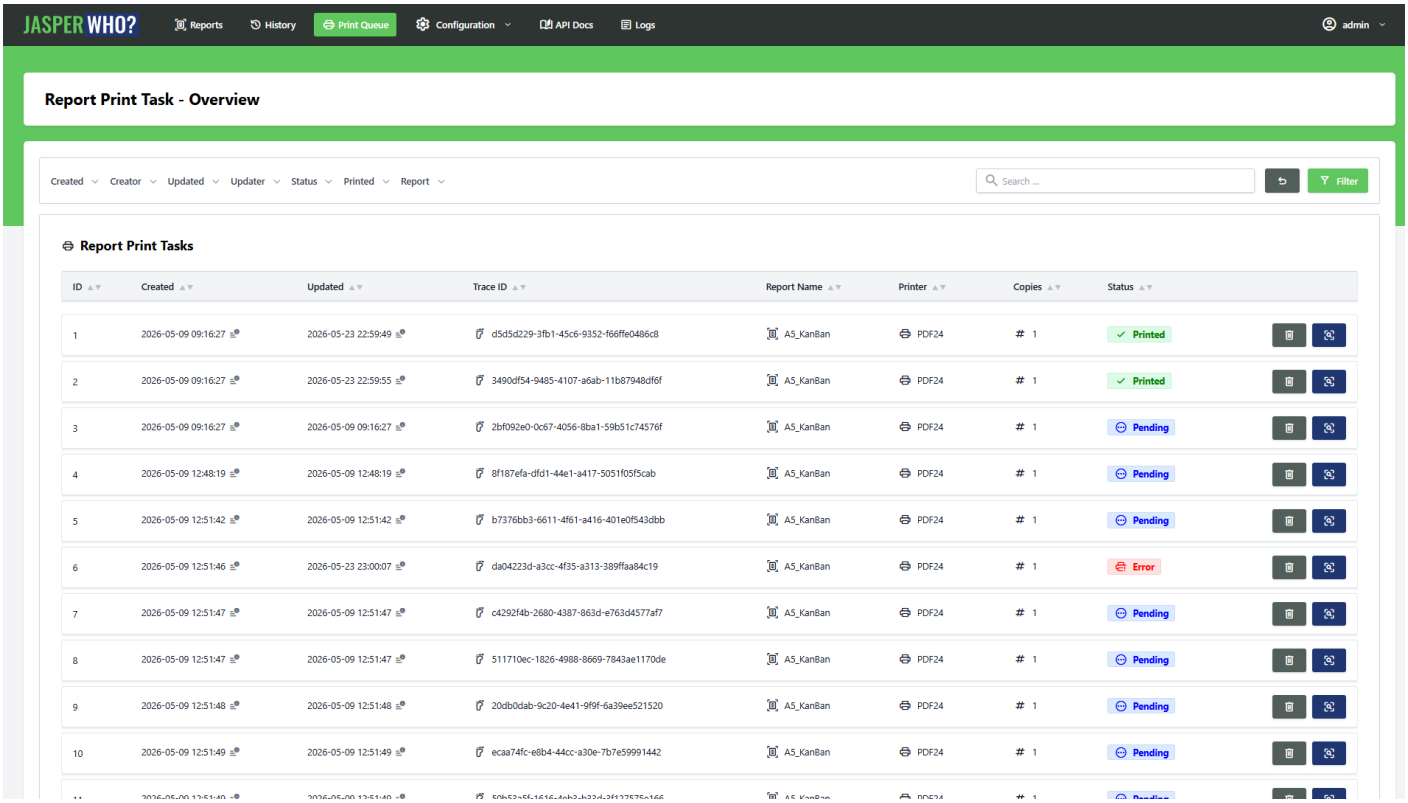
Creating Report Print Tasks

A `ReportPrintTask` represents a single print job — a PDF queued for delivery to a physical printer. JasperWho? does not communicate with printers directly. Instead, it creates the task record, optionally notifies a separate print service via WebSocket, and waits for the service to report back. This page covers how tasks are created, how the status lifecycle works, and how to handle retries.



Three Ways to Create a Print Task

1. As Part of a Render Request

The most common path: set `createPrintTask: true` in the render request body, provide a `printerName`, and JasperWho? renders the report and dispatches it to the printer in a single call. No second request needed.

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

```
{
                    "parameters":          {          "P_ARTICLE_NUMBER":          "456128
                            "data":               [              {              ...
}
```

2. From a History Record

A task can be dispatched from any existing `ReportHistoryRecord` — without re-rendering the report. JasperWho? uses the PDF stored in the history record and creates a new print task from it:

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

```
{


}
```

This is the standard reprint path. See The concept of Report History Records for details.

3. Standalone via the Print Task API

Print tasks can also be created directly — independently of any render or history record. The `POST /api/v1/report-print-task` endpoint accepts any PDF as a Base64 string, making it possible to use the JasperWho? print infrastructure for documents that were not produced by JasperWho? at all.

```
POST                                                                    /api/v1/report-print-task

{




}
```

`reportConfig` and `reportHistoryRecord` are both optional on this endpoint — the task is created without either relation if they are not provided.

The Data Model

Field	Description
`traceId`	Unique identifier. Shared with the linked history record when the task was created via a render request. For reprints, a derived trace ID is generated (original + short random suffix).
`reportConfig`	The `ReportConfig` the printed PDF was generated from. Optional — not present for standalone tasks.
`reportHistoryRecord`	The linked `ReportHistoryRecord`. Optional — not present for standalone tasks.
`printerName`	The name of the target printer, as the print service expects it.
`numberOfCopies`	Number of copies passed to the print service. JasperWho? always renders once — the print service is responsible for duplication. Defaults to `1`.
`broadcastId`	WebSocket channel ID. If set at creation time, JasperWho? broadcasts a `ReportPrintTaskCreated` event via Laravel Reverb. Omit to use polling instead.
`outputFileName`	The filename of the PDF queued for printing.
`status`	Current state of the task. See below.
`errorMessage`	Failure detail reported by the print service. `null` unless status is `error`.

Status Lifecycle

Every print task starts as `pending`. The print service picks it up, executes the job, and reports the result back to JasperWho? via the API:

Status	Set by	Meaning
pending	JasperWho?	Task created, waiting for the print service to pick it up.

Status	Set by	Meaning
printed	Print service	Print job executed and confirmed.
error	Print service	Print job failed. `errorMessage` contains the failure detail.
unknown	—	Status could not be determined.

The print service reports back using the dedicated status endpoint:

```
PATCH                                                    /api/v1/report-print-task/{id}/set-status

{

                                                                    "errorMessage":

}
```

Resetting to Pending

A task can be reset to `pending` using the set-printed shortcut endpoint — setting the status flag to `false`:

```
PATCH                                                    /api/v1/report-print-task/{id}/set-printed/false
```

This re-queues the task. If the task has a `broadcastId`, JasperWho? re-broadcasts the `ReportPrintTaskCreated` event immediately — notifying the print service to pick the task up again without polling. This is the standard retry mechanism for failed or stalled print jobs.

WebSocket vs. Polling

How the print service learns about a new task depends on whether a `broadcastId` is set.

With `broadcastId` — JasperWho? broadcasts a `ReportPrintTaskCreated` event via WebSocket (Laravel Reverb) the moment the task is created. The print service subscribes to the channel identified by `broadcastId` and reacts immediately. This is the recommended mode for real-time printing — the task reaches the printer within milliseconds of the render completing.

Without `broadcastId` — No broadcast is sent. The print service must poll `GET /api/v1/report-print-task?status=pending` at a regular interval and process any tasks it finds. This works fine for workflows where sub-second delivery is not required.

> ℹ **WebSocket delivery requires Laravel Reverb to be running.** If Reverb is down, task creation will fail with an error rather than falling back silently to polling. Use Supervisor to keep the Reverb process alive — the same Supervisor configuration that manages the Laravel queue worker should include a `php artisan reverb:start` program entry. See Installing JasperWho? for a reference configuration.

Retention and Deletion

Print tasks are automatically purged after a configurable number of days, set via the `PURGE_PRINTTASKS_DAYS` environment variable (default: 30 days). Purging runs as a scheduled background job — no manual action required.

Individual tasks can also be deleted directly via the API at any time:

```
DELETE                                                    /api/v1/report-print-task/{id}
```

There are no deletion constraints on print tasks themselves — they can always be removed. However, deleting a print task is a prerequisite for deleting the linked `ReportHistoryRecord`, which in turn must be cleared before a `ReportConfig` can be deleted. Automatic purging handles this chain in the background once retention periods expire.

Revision #4
Created 2026-05-10 10:45:14 UTC by Benjamin Fischer
Updated 2026-05-23 22:05:31 UTC by Benjamin Fischer