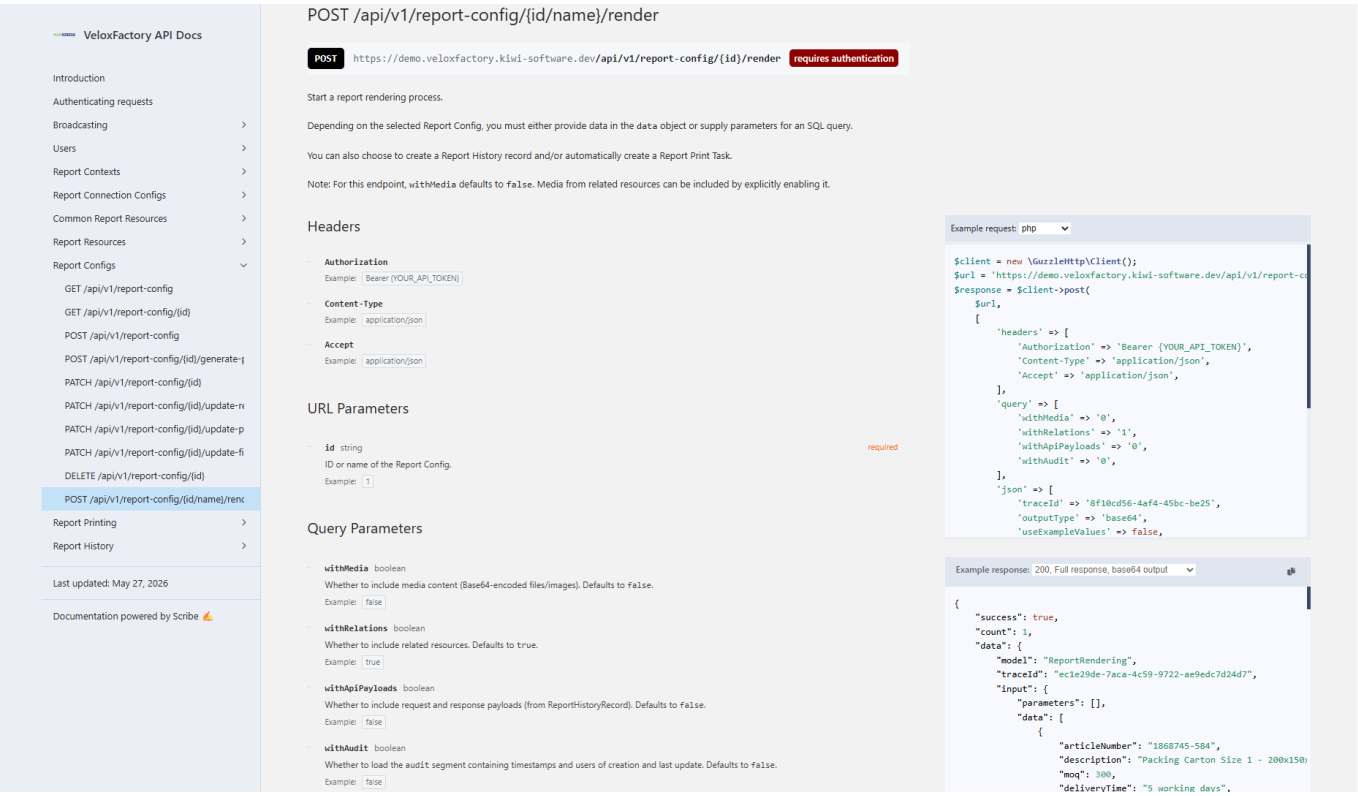
Rendering with our powerful API

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.



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.
`parameters`	object		Key-value map of parameter names to values. Required parameters must be present or the request is rejected.
`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.
`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.

Field	Type	Required	Description
`broadcastId`	string		WebSocket channel ID. If provided, VeloxFactory broadcasts a `ReportPrintTaskCreated` event when the print task is created. Omit to rely on polling.
`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.

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.

---

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
}
]
},
},
```

```
      ...
    },
    },
    }
  },
}
```

Laconic Response

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

```
{
```

```
    }
  },


}
```

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.

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 — empty data array for a report with a detail band, a type mismatch between field values and declared Java types, 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."
  ],









                                    {     "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.

Revision #5
Created 2026-05-10 10:44:35 UTC by Benjamin Fischer
Updated 2026-05-27 18:27:14 UTC by Benjamin Fischer