

Render engine: supported features and expressions

VeloxFactory renders `.jrxml` templates with its own render engine: a pure-PHP implementation of JasperReports, based on the open source project [JasperPHP](#) and maintained by us as an independent fork. It needs no Java runtime, but it implements a defined subset of JasperReports rather than the full Java engine. This page lists exactly what that subset is, so you can design templates in Jaspersoft Studio that render the same way in VeloxFactory.

How to set up parameters, fields, resources and the query is described in [Creating reports in Jaspersoft Studio](#). This page covers what happens inside the template: bands, elements, styles, expressions and formatting.

i This page describes render engine 1.2. If a template behaves differently from what is described here, the render engine is the reference, not Jaspersoft Studio's preview. Always check a real rendering in VeloxFactory before you release a template.

Two rules for every template

- **Every band needs** `splitType="Stretch"`, including an explicit empty `<background><band splitType="Stretch"/></background>`.
- **Bands must follow the JasperReports XSD order:** `background`, `title`, `pageHeader`, `columnHeader`, `detail`, `columnFooter`, `pageFooter`, `lastPageFooter`, `summary`. Jaspersoft Studio refuses to open a file with a different order.

Bands

Band	Behaviour
------	-----------

background	Printed once per page at the top edge, behind the content. Does not push other bands down.
title	Printed once on the first page. <code>{F{...}}</code> shows the first data row.
pageHeader	Printed on every page. On the first page it follows the title.
columnHeader	Printed on every page that contains detail rows, also after a manual <code><break></code> .
detail	Printed once per data row. Grows with stretching text fields.
groupHeader / groupFooter	Printed whenever the value of the <code>groupExpression</code> changes.
columnFooter	Printed below the detail rows of a page.
pageFooter	Printed at the bottom of every page.
lastPageFooter	Replaces the <code>pageFooter</code> on the last page. Also works without a <code>pageFooter</code> : then a footer appears on the last page only.
summary	Printed once at the end. <code>{F{...}}</code> shows the last data row, variables hold their final values. If the summary does not fit and moves to a page of its own, that page has no page header, column header or footer, and the <code>lastPageFooter</code> appears on the page before. With <code>isSummaryWithPageHeaderAndFooter="true"</code> on <code><jasperReport></code> , the summary page gets page header and footer (the <code>lastPageFooter</code>), but still no column header. This is JasperReports behaviour.
noData	Not supported.

Elements

Element	Supported
textField, staticText	Horizontal and vertical alignment, rotation, <code>markup="html"</code> , line spacing (Single, 1_1_2, Double, Proportional), <code>textAdjust</code> (StretchHeight, ScaleFont, CutText), <code><box></code> , <code>pattern</code> / <code>patternExpression</code> , hyperlinks
rectangle	<code>radius</code> , <code>mode</code> , forecolor and backcolor, pen width, colour and style (Solid, Dashed, Dotted), fractional sizes such as <code>height="0.8"</code>

Element	Supported
<code>line</code>	Pen as for rectangles. Lines thinner than 1 pt are drawn straight.
<code>frame</code>	Background, border, child elements positioned relative to the frame
<code>image</code>	Path or URL as an expression, Base64 data URIs (<code>data:image/png;base64,...</code>), JPG and PNG. In VeloxFactory, images normally come from <code>P_RESOURCE_</code> parameters.
Barcodes	QR Code, DataMatrix, PDF417, Code 128, Code 39, Code 93, EAN-8, EAN-13, Interleaved 2 of 5, Postnet. The code is an expression.
<code>subreport</code>	Data source from a list field (<code>\$F{items}</code>), also <code>new JRBeanCollectionDataSource(\$F{items})</code>), file name as an expression, <code>.jasper</code> is resolved to <code>.jrxml</code>
<code>break</code>	Page break, prints page footer, page header and column header
<code>jr:table</code>	Basic tables

Every element respects `printWhenExpression`, evaluated for each row.

Layout rules

- **StretchHeight** makes the band taller, never the elements beside it. The next band starts below the lowest stretched field.
- **Vertical alignment with StretchHeight:** if the text fits into the field, Top, Middle and Bottom apply as usual. Only overflowing text grows downwards from the top.
- `positionType="Float"` moves an element down by the growth of the stretched fields above it. All other elements keep their position.
- **ScaleFont** reduces the font size until the text fits into the box. Line breaks are allowed.
- **Page breaks** fill the page down to the page footer. The decision uses the declared band height.
- **<box>:** `<pen>` applies to all four sides, `topPen`, `leftPen`, `bottomPen` and `rightPen` override it. The colour falls back from the side pen to the box pen, then to `forecolor`, then to black. `padding` sets all four paddings, `leftPadding` etc. override it.

Defaults are the same as in JasperReports:

- Rectangle: Opaque, white background, 1 pt black border.
- Line: 1 pt. A pen width of 0 draws nothing.
- Pen colour: `lineColor` takes precedence over `forecolor`.

Styles

- **Assignment:** `style="Name"` on the `reportElement`. A style with `isDefault="true"` applies to all elements without a style of their own.
- **Inheritance:** `<style name="Child" style="Parent">`.
- **Precedence:** element attribute, then all matching conditional styles in order, then the style, then its parent.
- **Conditional styles** are evaluated again for every row, for example to colour negative quantities red or to shade every second row.

Styles carry: `mode`, `forecolor`, `backcolor`, horizontal and vertical text alignment (`hTextAlign`, `vTextAlign` and the older names), `rotation`, `markup`, font name, size, bold, italic, underline, strike-through, `pattern`, `radius`, image scaling and alignment, and the child elements `<pen>`, `<box>` and `<paragraph>`.

⚠ **An element with its own style does not inherit the default style.** This is JasperReports behaviour. If a style should build on the default style, set `style="Default"` as its parent.

Expressions

Expressions are written in Java syntax, as in Jaspersoft Studio. They are evaluated by a built-in interpreter; no code is executed on the server. Expressions work in text fields, `printWhenExpression`, conditional styles, variables, `patternExpression`, hyperlinks, images, barcodes and subreport file names.

Syntax

References	<code>\$P{name}</code> , <code>\$F{name}</code> , <code>\$V{name}</code>
Literals	<code>"text"</code> (with <code>\</code> \\ \n \t \uXXXX), <code>'text'</code> , <code>42</code> , <code>1.5</code> , <code>10L</code> , <code>2.5d</code> , <code>true</code> , <code>false</code> , <code>null</code>
Operators	<code>?</code> , <code>:</code> , <code> </code> , <code>&&</code> , <code>==</code> , <code>!=</code> , <code><</code> , <code><=</code> , <code>></code> , <code>>=</code> , <code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> , <code>!</code> , parentheses

Casts	(String), (Integer), (Long), (Double), (BigDecimal), (Number), (Boolean), (Date)
--------------	--

How values behave

- `+` concatenates as soon as one operand is text, from left to right: `1 + 2 + "a"` gives `3a`, `"a" + 1 + 2` gives `a12`.
- `7 / 2` gives `3` (integer division), `7 / 2.0` gives `3.5`.
- The type comes from the declared `class` of the field, parameter or variable. Declare amounts as `java.math.BigDecimal`, `java.lang.Double` or `java.lang.Integer`; a value declared as `String` is concatenated instead of added.

Three deliberate differences from Java make templates more robust:

- `null` is concatenated as empty text (Java: `"null"`).
- `==` compares values like `.equals()`.
- Arithmetic with `null` returns `null` instead of failing.

Methods

All values	<code>equals</code> , <code>toString</code> , <code>compareTo</code>
Text	<code>length</code> , <code>isEmpty</code> , <code>isBlank</code> , <code>trim</code> , <code>strip</code> , <code>toUpperCase</code> , <code>toLowerCase</code> , <code>substring</code> , <code>charAt</code> , <code>indexOf</code> , <code>lastIndexOf</code> , <code>contains</code> , <code>startsWith</code> , <code>endsWith</code> , <code>replace</code> , <code>replaceAll</code> , <code>replaceFirst</code> , <code>matches</code> , <code>concat</code> , <code>split</code> , <code>repeat</code> , <code>equalsIgnoreCase</code>
Numbers	<code>intValue</code> , <code>longValue</code> , <code>doubleValue</code> , <code>floatValue</code> , <code>setScale(n[, RoundingMode.X])</code> , <code>add</code> , <code>subtract</code> , <code>multiply</code> , <code>divide</code> , <code>negate</code> , <code>abs</code> , <code>signum</code> , <code>toPlainString</code>
Dates	<code>getTime</code> , <code>before</code> , <code>after</code>
Lists	<code>size</code> , <code>isEmpty</code> , <code>get(i)</code> , <code>contains</code>

Static calls, constants and constructors

- **Text:** `String.valueOf`, `String.format([Locale,] format, ...)`, `String.join`
- **Parsing:** `Integer.parseInt` / `valueOf`, `Long.parseLong`, `Double.parseDouble` / `valueOf`, `BigDecimal.valueOf`, `Boolean.valueOf` / `parseBoolean`
- **Math:** `abs`, `round`, `floor`, `ceil`, `max`, `min`, `pow`, `sqrt`
- **Constants:** `Boolean.TRUE` / `FALSE`, `BigDecimal.ZERO` / `ONE` / `TEN`, `RoundingMode.*`, `Locale.GERMANY`, `GERMAN`, `US`, `UK`, `ENGLISH`, `FRANCE`, `ITALY`
- **Constructors:** `new java.util.Date()`, `new Date(ms)`, `new java.text.SimpleDateFormat("...", [Locale])` with `.format()` and `.parse()`, `new`

```
java.text.DecimalFormat("...").format(), new BigDecimal("1.5"), new Integer(...), new  
String(...), new Locale("de", "DE")
```

Examples

```
"Customer:                                "                                +                                ${P_CUSTOMER}  
${P_NOTE} == null || ${P_NOTE}.isEmpty() ? "-" : ${P_NOTE}  
${F{status}.equals("late")} && ${F{qty}} > 0  
String.format("%s / %05d", ${F{order}}, ${F{pos}})  
new java.text.SimpleDateFormat("EEEE, d. MMMM yyyy", java.util.Locale.GERMAN).format(${F{date}})  
${F{amount}}.setScale(2, java.math.RoundingMode.HALF_UP)
```

Legacy suffixes

Two JasperPHP shortcuts still work for existing templates: `${F{amount}.toString()}` always returns two decimals in German format (`1.234,50`), and `${F{amount}.numberToText()}` spells out a number (Portuguese). For new templates, use `pattern` instead.

Not supported

`$R{...}` (resource bundles), arbitrary Java classes, lambdas and field access without a method call (`.length` without parentheses). A text field with an unsupported expression falls back to plain placeholder substitution, which usually shows parts of the expression in the output. A condition with an unsupported expression counts as false, so the element is hidden.

Formatting numbers and dates

- **pattern** / **patternExpression**: `patternExpression` takes precedence and is evaluated for each row, for example to switch the number of decimals per article.
- **Numbers** use Java's `DecimalFormat`: `#,##0.00`, `0.0%`, `000000`, `#,##0.00 €`, `#,##0.00;(#,##0.00)`, `0.00E0`. Rounding is `HALF_EVEN` as in Java; `setScale` rounds `HALF_UP` by default.
- **Dates** use Java's `SimpleDateFormat`: `dd.MM.yyyy`, `EEEE, d. MMMM yyyy`, `HH:mm`.
- **Values that arrive as text** are formatted as well when they contain a number or an ISO date (`2026-05-02`, `2026-05-02 14:30:00`). This matters because render requests usually pass parameters as text.

Language and locale

Pass the parameter `REPORT_LOCALE` in the render request, for example `de_DE`. It controls decimal and thousands separators, month and day names, `String.format` and `pattern`. Built in are German (including `de_CH`), English, French, Spanish, Italian and Portuguese. Without `REPORT_LOCALE`, the default locale of the installation applies (English unless configured otherwise).

i Do not declare `REPORT_LOCALE` as a parameter in the template. It is a built-in JasperReports parameter, and Jaspersoft Studio rejects a second declaration. Pass it in the render request or the schedule only.

Variables

- **Calculations:** Sum, Count, Average, DistinctCount, Highest, Lowest, First, Nothing (current value). Count and Average only count values that are not null.
- **Reset:** `resetType="Report"` or `"Group"`. Groups reset correctly even without a group header band.
- **Expressions** can be any supported expression, for example `$F{qty} * $F{price}`.
- **Built in:** `PAGE_NUMBER` and `REPORT_COUNT`. For "Page N of M", use two text fields: `"Page " + $V{PAGE_NUMBER}`, and `" of " + $V{PAGE_NUMBER}` with `evaluationTime="Report"`.

Upgrading templates from earlier versions

Templates built for earlier VeloxFactory versions keep working. Check them for these changes:

- **Conditions in PHP syntax** (for example `is_int(...)`, `===`, `and` or `or` in `printWhenExpression`) now count as false, and the element is hidden. Rewrite them in Java syntax.
- **Lines without `<pen>`** are 1 pt wide instead of a hairline.
- **Element attributes override the style** (previously the other way round).
- **Multi-page reports** fit more rows per page and repeat the column header on every page.
- **Workarounds are no longer needed:** concatenation with `+`, `pattern`, `<box>` and page footers on every page now work as in Jaspersoft Studio.

Known limits

- Charts and crosstabs are not supported and render as an empty area. Use tiles, lists and aggregated groups instead.
- `noData` band and `$R{...}` are not supported.
- `isBlankWhenNull="false"` is ignored; null always prints as empty text.
- A band that stretches shortly before the end of a page can reach into the page footer. Leave some space above the footer on pages with long texts.

Revision #3

Created 2026-09-28 19:45:08 UTC by Benjamin Fischer

Updated 2026-09-28 19:58:16 UTC by Benjamin Fischer