Amtiri

REFERENCE MANUAL

Shapes and text

Generated on November 25, 2026

Six widgets draw the texts, lines and shapes a plant screen is made of: tanks, pipes, pumps, their states and their readings. All six are SVG shapes on the dashboard's canvas. They are drawn in the order of the widget list, the first widget on top; they take SVG style keys such as `fill` and `stroke`; and they look the same in the viewer's light and dark themes.

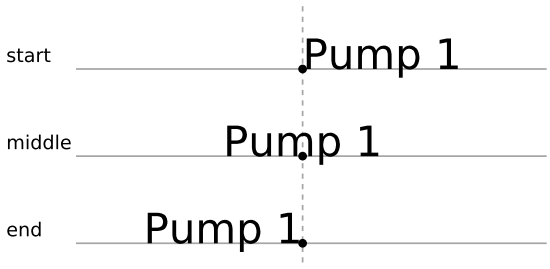
Widget	Palette tile	Stored type	Draws	Typical use
Label	Text	`Label`	One line of text	A reading, a tag, a title
Distributed labels	Distributed Labels	`Labels`	Several texts spread evenly along a line	Scale marks, axis captions
Rectangle	Rectangle	`Rect`	A rectangle filling its box	Tanks, level bars, frames
Circle	Circle	`Circle`	The largest circle that fits its box	Status lamps, pump symbols
Ellipse	Ellipse	`Ellipse`	An ellipse filling its box	Vessel ends, valve bodies
Line	Line	`Line`	A straight segment between two points	Pipes, arrows, dividers

Every widget also has the properties all widgets share: its name, its position and size, `visible`, `styles`, and the offsets `dX`, `dY`, `dWidth` and `dHeight`. They are described once, in Properties every widget shares, and the rules of the `styles` record in Styling widgets. The tables below list only each widget's own properties.

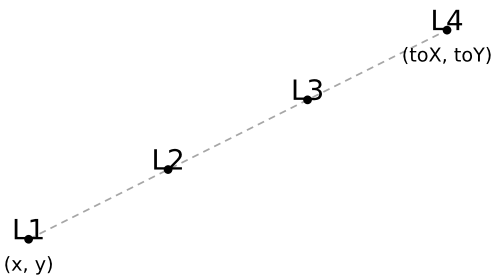
Label

A label draws one line of text, the result of its `label` formula, at its point `(x, y)`.

Label: the point is on the baseline



Distributed labels: item i of n at i / (n − 1)



The dot is (x, y); text_anchor decides which part of the text sits on it. Four labels sit at 0, 1/3, 2/3 and 1 of the way, each standing c

- **The point is on the baseline.** The text stands on it as on a ruled line: the letters rise above the point, and descenders such as "g" and "p" drop below it. `text_anchor` in the styles decides which part of the text sits on the point: its start, its middle (the default) or its end. A column of readings aligned on their right edge is a column of labels with the same `x` and `text_anchor:"end"`.
- **One line.** The text is drawn on one line, and never wraps. Use one label per line.

- **Text or a number.** `label` normally returns text. A number is shown as its text: `42.5` shows `42.5` and `3` shows `3`, but arithmetic gives decimals, so `2 + 1` shows `3.0`. Format a reading for a steady look: `"%.1f".format(LIT_301)` always shows one decimal (see `format`).
- **A formula that fails**, for example on a variable that was never written, shows no text.
- **Size.** Without `font_size` in the styles the text is 12 px.
- **Alignment without styles.** `horizontalAlignement`, spelt so in the stored record, holds `LEFT`, `CENTER` or `RIGHT` and has no editor. It aligns the text only when the styles give no `text_anchor`. It also decides where the editor puts the label's click area around the point, so a label whose styles say `text_anchor:"start"` is still picked as if its text were centred. Align with `text_anchor`.
- **In the editor** a label has one handle, on its point. The click area around the text is estimated from the font size and the number of characters, so it can be wider or narrower than the text.
- **Offsets.** `dX` and `dY` move the point in the viewer.

Properties

Key	Editor label (English)	Formula or value	Result type	Default	Meaning
`label`	Label	Formula; the editor evaluates it too	Text; a number shows as its text	`"Text"`	The text to show
`horizontalAlignement`	None: no editor	Value	`LEFT`, `CENTER` or `RIGHT`	`CENTER`	The alignment used when the styles give no `text_anchor`; also where the editor's click area lies

Example

A tank level with its unit, in red above 4.2 m:

```
Label    "%.2f".format(LIT_301) + " m"
Styles   {fill:if(LIT_301 > 4.2,"#B91C1C","#111827"),font_size:"24px",font_weight:"bold",text_anchor:"end"}
```

With `LIT_301` at 3.4567 the label shows `3.46 m` in near-black, its right edge on the point.

Styling

- **Element**: one SVG `<text>`. `styles` applies to it.
- **Default styles**: `{fill:"#000",font_size:"32px",text_anchor:"middle"}`.
- **Keys that work**: `fill` (the colour of the letters), `stroke` and `stroke_width` (an outline around them), `font_family`, `font_size`, `font_weight` (300 to 800 with Open Sans), `font_style`, `letter_spacing`, `text_anchor` and `opacity`.
- **Keys to avoid**: `transform`, which works from the canvas's top-left corner rather than from the label's point.
- **Sizes**: a number for `font_size` or `letter_spacing` is written in pixels; a text must carry its unit, as `"24px"` does.
- **Fonts**: Open Sans is the only font every viewer has. End any other family with a generic one: `font_family:"'Courier New', monospace"`.

The widget reads two style keys itself:

Key	What the widget does with it
`font_size`	Sizes the text. The editor also sizes the label's click area from it. Without it, the text is 12 px
`text_anchor`	`start`, `middle` or `end`: the part of the text that sits on the point. It takes the place of `horizontalAlignement`

Styled example. A grey caption in a monospaced face, starting at its point:

```
Styles   {fill:"#4B5563",font_size:"14px",font_family:"'Courier New', monospace",text_anchor:"start"}
```

Distributed labels

Distributed labels draw several texts, the items of the `labels` list, spread evenly along the segment from the start point `(x, y)` to the end point `(toX, toY)`. Item i of n sits i/(n # 1) of the way: the first on the start point, the last on the end point, and the others at equal steps between them, as the drawing under Label shows. A list of one item puts it on the start point. Each text stands on its point as a label's does, centred by default.

- **The list** holds texts or numbers: `["0", "1.5", "3", "4.5"]` and `[0, 1.5, 3, 4.5]` show the same four texts. Numbers show as their text, as on a Label. When the list changes length, the texts are spread again.
- **A formula that fails** draws no text at all.
- **Both points are absolute** canvas positions. In the editor the widget has three handles: the start point, the end point, and the middle, which moves both. `toX` and `toY` have no field: drag the end handle. A new widget runs from its start 80 px to the right and 32 px up, centred on the point where it was dropped.
- **Offsets.** In the viewer `dX` and `dY` move the start point and `dToX` and `dToY` move the end point, so moving the whole widget takes the same formulas on both ends.
- **One style for every item.** All the texts get the same styles, so one item cannot be coloured apart from the others: use separate Labels for that.
- **Size.** Without `font_size` in the styles the texts are 12 px.

Properties

Key	Editor label (English)	Formula or value	Result type	Default	Meaning
`labels`	Labels	Formula; the editor evaluates it too	List of texts or numbers	`["L1", "L2", "L3", "L4"]`	The texts to spread, first to last
`toX`	None: the end handle	Value	Number, canvas px	Start + 80	Horizontal position of the end point
`toY`	None: the end handle	Value	Number, canvas px	Start # 32	Vertical position of the end point
`dToX`	To X offset	Formula, viewer only	Number, px	Empty	Added to `toX`. Empty or failing counts as 0
`dToY`	To Y offset	Formula, viewer only	Number, px	Empty	Added to `toY`. Empty or failing counts as 0

Example

The scale beside a 4.5 m tank, drawn from the start point at the tank's bottom, `(500, 300)`, to the end point at its top, `(500, 120)`:

```
Labels   ["0 m", "1.5 m", "3 m", "4.5 m"]
Styles   {fill:"#374151",font_size:"12px",text_anchor:"end"}
```

The four marks stand at y = 300, 240, 180 and 120, their right edges on x = 500.

Styling

- **Element**: an SVG `<g>` holding one `<text>` per item. `styles` applies to every `<text>` alike.
- **Default styles**: `{fill:"#000",font_size:"14px",text_anchor:"middle"}`.
- **Keys that work**: as on a Label: `fill`, `stroke`, `stroke_width`, `font_family`, `font_size`, `font_weight`, `font_style`, `letter_spacing`, `text_anchor` and `opacity`.
- **Keys to avoid**: `transform`, as on a label. The editor's click areas assume centred texts, whatever `text_anchor` says.

The widget reads two style keys itself:

Key	What the widget does with it
`font_size`	Sizes the texts, and the editor's click areas around them. Without it, the texts are 12 px
`text_anchor`	`start`, `middle` or `end`: the part of each text that sits on its point

Styled example. Hour captions under a chart, in small grey type:

```
Labels   ["00:00", "06:00", "12:00", "18:00", "24:00"]
Styles   {fill:"#6B7280",font_size:"11px",text_anchor:"middle"}
```

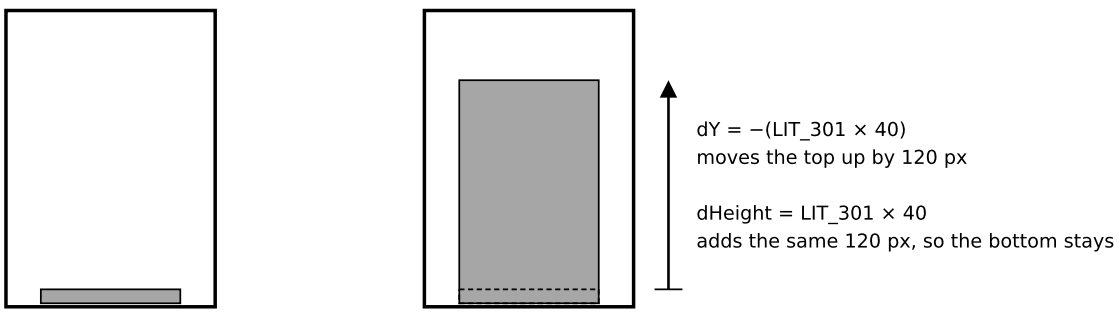
Rectangle

A rectangle fills its box: from `(x, y)`, `width` wide and `height` high. A new one is 80 × 32 px, centred on the point where it was dropped, with four corner handles and a middle handle that moves it. Round its corners with `rx` and `ry` in the styles.

A rectangle has no properties of its own: it has only the properties every widget shares.

Level bar

A bar that rises with a level is a rectangle whose offsets grow it upwards. `dHeight` adds the level, in pixels, to the height, and `dY` moves the top edge up by the same amount, so the bottom edge stays where it was drawn:



In the editor: the bar as drawn

In the viewer, with LIT_301 = 3

Draw the bar at the bottom of the tank with the height it has when the tank is empty: a few pixels are enough to find it in the editor, which always shows the box as drawn. Then write the offsets, here with 40 px per metre for a tank of 4.5 m, and `clamp` to keep the bar inside the tank:

```
dY        -(clamp(LIT_301, 0, 4.5) * 40)
dHeight   clamp(LIT_301, 0, 4.5) * 40
Styles    {fill:if(LIT_301 > 4.2,"#DC2626","#38BDF8"),stroke:"none"}
```

An offset that fails counts as 0, so a level that was never written shows the bar as drawn. For a bar that grows to the right, use `dWidth` alone: the left edge stays.

Styling

- **Element**: one SVG `<rect>`. `styles` applies to it.
- **Default styles**: `{fill:"#6CF",stroke:"#000",stroke_width:3}`.
- **Keys that work**: `fill`, `fill_opacity`, `stroke`, `stroke_width`, `stroke_opacity`, `stroke_dasharray` (as text, `"8 4"`), `stroke_linejoin`, `rx` and `ry` (rounded corners), `opacity` and `cursor`. The stroke is drawn centred on the box's edge, half of it outside the box.
- **Keys to avoid**: `x`, `y`, `width` and `height`, which some browsers take as the rectangle's geometry and draw instead of its box.
- `rx` and `ry` are lengths: a number is written in pixels, and a text must carry its unit.

The widget reads no style key itself.

Styled example. A frame with rounded corners and a dashed border while a unit is out of service:

```
Styles   {fill:"#F3F4F6",stroke:"#6B7280",stroke_width:2,rx:8,ry:8,stroke_dasharray:if(UNIT_2_OOS,"6 4","none")
```

Circle

A circle is the largest circle that fits its box: its diameter is the box's shorter side, and it is centred in the box. Widening the box does not stretch it; use an Ellipse for that. A new circle is 48 × 48 px, centred on the point where it was dropped.

A circle has no properties of its own: it has only the properties every widget shares.

Styling

- **Element**: one SVG `<circle>`. `styles` applies to it.
- **Default styles**: `{fill:"#6CF",stroke:"#000",stroke_width:3}`.
- **Keys that work**: `fill`, `fill_opacity`, `stroke`, `stroke_width`, `stroke_opacity`, `stroke_dasharray`, `opacity`, `cursor`, and an `animation` that names one of the dashboards' keyframes, `do-blink` or `do-spin`.
- **Keys to avoid**: `r`, `cx` and `cy`, which some browsers take as the circle's geometry and draw instead of its box.

The widget reads no style key itself.

Styled example. A pump lamp: green when running, grey when stopped, and red and blinking when tripped:

```
Styles   {fill:if(P_101_TRIP,"#DC2626",if(P_101_RUN,"#16A34A","#9CA3AF")),stroke:"#374151",stroke_width:2,anima
```

A viewer whose system asks for reduced motion sees no blinking, so the colour carries the alarm on its own. The keyframes are described in Styling widgets.

Ellipse

An ellipse fills its box: its horizontal radius is half the box's width, its vertical radius half its height, and it is centred in the box. A new one is 80 × 32 px, centred on the point where it was dropped.

An ellipse has no properties of its own: it has only the properties every widget shares.

Styling

- **Element**: one SVG `<ellipse>`. `styles` applies to it.
- **Default styles**: `{fill:"#6CF",stroke:"#000",stroke_width:3}`.
- **Keys that work**: `fill`, `fill_opacity`, `stroke`, `stroke_width`, `stroke_opacity`, `stroke_dasharray`, `opacity` and `cursor`.
- **Keys to avoid**: `rx`, `ry`, `cx` and `cy`. They are the ellipse's own geometry, and some browsers draw them instead of its box.

The widget reads no style key itself.

Styled example. The end of a horizontal vessel, shaded while it is filling:

```
Styles   {fill:if(TK_12_FILLING,"#BAE6FD","#E5E7EB"),stroke:"#4B5563",stroke_width:2}
```

Line

A line runs from its start point `(x, y)` to its end point `(toX, toY)`. In the editor it has three handles: the start, the end, and the middle, which moves both. A new line runs from its start 80 px to the right and 24 px up, centred on the point where it was dropped.

- **Both points are absolute** canvas positions. `toX` and `toY` have no field: drag the end handle.
- **Offsets.** In the viewer `dX` and `dY` move the start point and `dToX` and `dToY` move the end point. A pointer that swings with a value moves one end only.
- **Ends are round.** The canvas draws every line end round, which joins pipes drawn as separate lines without gaps. `stroke_linecap:"butt"` cuts the ends square at the points.

Properties

Key	Editor label (English)	Formula or value	Result type	Default	Meaning
`toX`	None: the end handle	Value	Number, canvas px	Start + 80	Horizontal position of the end point
`toY`	None: the end handle	Value	Number, canvas px	Start # 24	Vertical position of the end point
`dToX`	To X offset	Formula, viewer only	Number, px	Empty	Added to `toX`. Empty or failing counts as 0
`dToY`	To Y offset	Formula, viewer only	Number, px	Empty	Added to `toY`. Empty or failing counts as 0

Styling

- **Element**: one SVG `<line>`. `styles` applies to it.
- **Default styles**: `{stroke:"#000",stroke_width:3}`.
- **Keys that work**: `stroke`, `stroke_width`, `stroke_opacity`, `stroke_dasharray` (as text, `"10 6"`), `stroke_linecap` (`round` by default, `butt` or `square`) and `opacity`.
- **Keys to avoid**: `fill`, which a line does not use.

The widget reads no style key itself.

Styled example. A pipe that is blue while its valve is open, and grey and dashed while it is closed:

```
Styles   {stroke:if(FV_201_OPEN,"#2563EB","#9CA3AF"),stroke_width:6,stroke_dasharray:if(FV_201_OPEN,"none","10
```

Next steps

- How dashboards draw and refresh
- Properties every widget shares
- Styling widgets
- Gauges
- Groups

This page describes Data Orchester Dashboards 1.9.4.