Amtiri

REFERENCE MANUAL

Styling widgets

Generated on November 25, 2026

A widget's look is a formula. Its `styles` returns a record of style keys, and the widget writes them onto what it draws: an SVG shape or text on the canvas, or an HTML control over it. Because the record is computed, the look can follow the plant: a pump turns green when it runs, a level turns red above its alarm. This page gives the rules of the record, colours and units, what each widget's element accepts, the theme, animation, and a set of recipes. Each widget's own page adds the element its styles land on, its default styles and the keys to avoid there.

The styles record

```
{fill: if(P101_RUN, "#16A34A", "#9CA3AF"), stroke: "#374151", stroke_width: 2}
```

- **Keys are names,** written without quotes. Every `_` in a key is written as `-`: `stroke_width` sets the property `stroke-width`, and `background_color` sets `background-color`. A key that starts with two underscores sets a CSS custom property: `__j2w_accent` sets `--j2w-accent`.
- **Values are written as text,** exactly as they are. The browser keeps what it understands and drops the rest silently: a misspelt key or an unusable value fails on its own, and nothing else does.

Value in the record	Written as	Result
Text: `fill: "#2563EB"`	`fill: #2563EB`	Used
Whole number: `stroke_width: 3`	`stroke-width: 3`	Used
Decimal: `opacity: 0.5`	`opacity: 0.5`	Used
A number for a length: `font_size: 18`	`font-size: 18px`	Used: see Units
Boolean: `fill: true`	`fill: true`	Dropped by the browser: a fill turns black
List: `stroke_dasharray: [8, 4]`	The list as text	Dropped: write the text `"8 4"`
Record: `fill: {r: 1}`	The record as text	Dropped
`null`	Nothing	The key is removed

- **Each redraw replaces the whole set.** A key that the new record leaves out is removed, and so is a key whose value is `null`: the property returns to what the widget or the browser gives it. So a key needed in one state only can be left out, or be `null`, in the others. `{fill: "#E5E7EB", stroke: if(ALARM, "#DC2626", null), stroke_width: 4}` draws a red outline during the alarm and none after it.
- **A styles formula that fails keeps the last styles that worked.** One that has never worked leaves the element with the browser's own look: an SVG shape is then filled black, with no outline. One that does not compile stops its widget from being updated at all: see When a formula fails.
- **Styles are evaluated in the editor too,** so a dashboard shows its colours while it is built.

- **The gauges take a second record, `valueStyles`,** for their value: the arc of a RadialGauge, the bar of a HorizontalGauge or a VerticalGauge. A RadialGauge's `styles` draws its track, the whole arc behind the value, light grey by default. The linear gauges do not apply `styles`: draw their track with a Rect placed after the gauge in the component list.
- **Two widgets take no styles at all:** the SVG widget, whose drawing keeps its own look, and a group, whose members are styled one by one.

Keys the widgets read themselves

Most keys only reach the browser. A few are also read by the widget:

Key	Widgets	What the widget does with it	When it is left out
`font_size`	Label, Labels, RadialLabels	Sizes the text, and the editor's estimate of the area a click on the text selects	12 px
`text_anchor`	Label, Labels, RadialLabels	Which point of the text sits on its anchor: `start`, `middle` or `end`	`middle`, or a Label's stored alignment
`bar_width`	ChartBar	The width of a bar, as a fraction of the slot each value has, given as a number or numeric text	0.9
`bar_offset`	ChartBar	The gap before a bar, as a fraction of its slot, given as a number or numeric text	0.05
`r`	ChartDot	The radius of a dot in canvas pixels, given as a number or numeric text such as `"6px"`	6
`background_true`	FieldSwitch	The track's colour while the switch is on	The theme's accent colour
`background_false`	FieldSwitch	The track's colour while it is off	The theme's sunken surface
`cursor`	Region	The pointer's shape over the region	`pointer`, a hand

Colours

A colour in a style is text, in any form a browser reads:

Form	Example	Notes
Hex, three or six digits	`"#6CF"`, `"#2563EB"`	An opaque colour
Hex, eight digits	`"#DC262680"`	The last pair is the opacity, from `00`, transparent, to `FF`: `80` is about half
Name	`"orange"`, `"transparent"`	Any CSS colour name
CSS functions, as text	`"rgb(37 99 235)"`, `"hsl(217 91% 60%)"`	Written in quotes, like any other text
`none`	`"none"`	No fill, or no outline, on an SVG widget

The colour functions

The colour functions of Amtiri Script return a **colour number**, not text: `rgb(1, 0, 0)` is 16711680. A style writes that number as it is, the browser drops it, and a fill turns black. Turn it into hex text with `format` and the pattern `%06X` (hexadecimal, upper case, six digits): `"#%06X".format(rgb(1, 0, 0))` gives `"#FF0000"`.

Function	Takes	Example	Style text
`rgb(r, g, b)`	Red, green and blue, each a fraction from 0 to 1. 1 and above count as 1, so `rgb(128, 64, 0)` is yellow	`"#%06X".format(rgb(1, 0.5, 0))`	`#FF8000`
`hsl(h, s, l)`	The hue in turns, from 0 to 1: 0 red, 1/3 green, 2/3 blue, and 1 red again. Saturation and lightness from 0 to 1	`"#%06X".format(hsl(0.3333, 1, 0.5))`	`#00FF00`
`lighter(colour, f)`, `darker(colour, f)`	A colour number, and a factor from 0 to 1	`"#%06X".format(darker(rgb(1, 0, 0), 0.5))`	`#602020`
`gradient(name, value)`	A gradient's name, written bare, and a value from 0 to 1	`"#%06X".format(gradient(red2green, 0.5))`	`#FFFF00`

- **Gradients.** The engine offers seven: `red2green`, `red2gray2green`, `red2white2green`, `black2green`, `black2red`, `white2black`, and `rainbow`, which runs from red through yellow, green and blue back to red. The name is bare and its case does not matter; a name the engine does not have makes the formula fail when it runs, and the widget keeps its last styles. A value below 0 gives the gradient's first colour, and one above 1 its last.
- **A colour that follows a reading** divides the reading by its range: `{fill: "#%06X".format(gradient(red2green, 1 - to_double(LIT_301) / 4.5))}` is green when the tank is empty and red when it is full.

> **Warning**
>
> `format` writes decimals the way the instance's locale does: on an instance whose locale uses a decimal comma, `"%.2f".format(0.5)` gives `0,50`, which no browser reads. Give numbers to a style as numbers, as in `opacity: LOAD_PCT / 100`, and keep `format` for colours (`%06X`) and whole numbers (`"%dms".format(to_int(X))`).

Units

Sizes are in canvas pixels, and scale with the rest of the drawing when the view is zoomed.

- **A number given for a length is written with `px`.** The keys are `font_size`, `letter_spacing`, `word_spacing`, `rx`, `ry`, `border_radius`, `border_width`, `outline_width`, `padding`, `min_height` and `min_width`: `font_size: 18` becomes `font-size: 18px`, and `padding: 6` pads all four sides by 6 px. A text with its unit, such as `"18px"`, works for them too.
- **A number for any other key is written as it is.** That is what `stroke_width`, `opacity`, `fill_opacity`, `stroke_opacity` and `font_weight` expect.
- **Everything else is text, with its unit:** `border: "2px solid #DC2626"`, `padding: "4px 12px"`, `box_shadow: "0 2px 6px #0000004D"`, `stroke_dasharray: "8 4"`, `transition: "fill 0.5s"`.

SVG widgets and HTML widgets

A widget draws either on the canvas, as SVG, or over it, as an HTML control, and the two take different properties:

- **SVG widgets**, the shapes, texts, images, gauges, charts and regions, take SVG properties: `fill` for the inside, `stroke` for the outline, `font_family` and its kin for text. They do not change with the viewer's theme.
- **HTML widgets**, the seven inputs and the two buttons, take CSS box properties: `background_color`, `color` for the text, `border`, `border_radius`, `padding`. They follow the viewer's theme until their styles say otherwise.

A property of the other kind does nothing: `fill` on a button, `background_color` on a Rect.

Widget	The styles land on	Properties that work there	Keys to avoid
Label	An SVG text, standing on its baseline	`fill`, `stroke` with `stroke_width` for an outline, `font_family`, `font_size`, `font_weight`, `font_style`, `letter_spacing`, `text_anchor`, `opacity`	`transform`
Labels	An SVG text per item, all styled alike	As Label	`transform`
RadialLabels	An SVG text per item, each turned by the widget	As Label	`transform`, which replaces the turn of every label
Rect	An SVG rectangle	`fill`, `fill_opacity`, `stroke`, `stroke_width`, `stroke_opacity`, `stroke_dasharray`, `stroke_linejoin`, `rx` and `ry` for round corners, `opacity`	`x`, `y`, `width`, `height`, which some browsers let replace the drawn box
Circle	An SVG circle, as wide as the box's shorter side	As Rect, without `rx` and `ry`	`r`, `cx`, `cy`
Ellipse	An SVG ellipse filling the box	As Circle	`rx`, `ry`, `cx`, `cy`
Line	An SVG line, with round ends unless `stroke_linecap` says otherwise	`stroke`, `stroke_width`, `stroke_opacity`, `stroke_dasharray`, `stroke_linecap`, `opacity`	`fill`, which has nothing to fill
Image	An SVG image	`opacity`, `cursor`	—
SVG	Nothing: its styles are not applied	—	Every key. Hide or move it with `visible` and the offsets
RadialGauge	`styles`: the track, an SVG polyline along the whole arc. `valueStyles`: the value arc, a second polyline	`stroke`, `stroke_width`, `stroke_linecap`, `stroke_dasharray`, `stroke_opacity`, `opacity`	A `fill` other than `"none"`, which fills the arc's chord
HorizontalGauge, VerticalGauge	`valueStyles`: the bar, an SVG rectangle. `styles` is not applied	As Rect	—
ChartLine	One SVG polyline through the values	`stroke`, `stroke_width`, `stroke_dasharray`, `stroke_linejoin`, `stroke_linecap`, `opacity`, and `fill: "none"`	—
ChartArea	One SVG polyline, closed down to the bottom of the box	`fill`, `fill_opacity`, `stroke`, `stroke_width`, `opacity`	—
ChartBar	An SVG rectangle per value	As Rect, and `bar_width`, `bar_offset`	—
ChartDot	An SVG circle per value	`fill`, `stroke`, `stroke_width`, `opacity`, and `r`	—

ChartGridH, ChartGridV	An SVG line per value	As Line	—
Region	An SVG rectangle, almost transparent	`cursor`, and a visible `fill` while placing it	`fill: "none"`, `visibility: "hidden"` and `display: "none"`: a region that is not painted cannot be clicked
FieldButton, ActionButton	An HTML button	`background_color`, `color`, `border`, `border_color`, `border_width`, `border_radius`, `padding`, `font_family`, `font_size`, `font_weight`, `box_shadow`, `opacity`, `min_height`, `cursor`	`fill` and `stroke`, which do nothing there. `position`, `left`, `top`, `width` and `height`, which replace the place and size the widget was drawn at
FieldText	An HTML text box, or a text area when **Multiline** is ticked	As a button, and `text_align`	As a button
FieldNumber	An HTML number box, aligned right	As FieldText	As a button
FieldDate	An HTML date box	As FieldText	As a button
FieldCombobox	An HTML drop-down list. The list that opens is drawn by the browser	As FieldText	As a button
FieldSliderHorizontal, FieldSliderVertical	An HTML range slider. The vertical one is the same slider, turned	`__j2w_accent` for the filled part and the thumb, `opacity`	`transform`, which replaces the vertical slider's turn. The slider's own `__j2w_slider_...` properties, which it sets itself
FieldSwitch	An HTML switch	`background_true`, `background_false`, `opacity`	`background_color`, which competes with the two state colours
ComponentGroup	Nothing: its styles are not applied	—	Every key. Style the members

Every widget that takes styles also takes the keys of the sections below: `cursor`, `transition`, `animation` and `display`.

Fonts

- **Open Sans is the only font the product serves,** in every weight from 300 to 800, upright and italic. Texts and controls use it unless their styles name another family.
- **Any other family comes from the viewer's device.** A device that lacks it shows the browser's default font instead, often a serif. End every `font_family` with a generic family, which every device has: `sans-serif`, `serif`, `monospace` or `system-ui`. A family name with spaces goes in single quotes inside the text: `font_family: "'Courier New', monospace"`.
- `font_weight` takes a number, any from 300 to 800 with Open Sans (`600` is semibold), or `"normal"` and `"bold"`. `font_style: "italic"` slants the text.

The theme

- **Light and dark.** Each user chooses a light or a dark theme, or follows the device. The theme changes the HTML widgets only: in dark mode the inputs turn dark and the buttons change tint, while SVG shapes, texts and the canvas background keep their colours. A dashboard drawn on a white canvas therefore shows dark inputs in dark mode. Where that matters, give inputs and buttons their own `background_color`, `color` and `border_color`: they then look the same in both themes.
- **A widget's styles win over the theme,** in both themes.
- **Hover, focus and pressed looks cannot be styled.** A style applies to the widget in every state. The theme tints a button while the pointer is over it and while it is pressed; setting `background_color` removes both tints, and only a slight shrink on press remains, so give such a button `cursor: "pointer"`. An input keeps the theme's hover and focus border colours unless `border_color` is set. SVG shapes have no hover look beyond `cursor`.
- **One custom property is meant for widgets:** `__j2w_accent`, the accent colour. On a slider it paints the filled part of the track and the thumb, which no other key reaches: `__j2w_accent: "#DC2626"`. A switch has its own two keys, `background_true` and `background_false`, which default to the theme's colours when they are left out.
- **A brand sheet restyles every dashboard at once.** The instance setting `application.theme` names an extra stylesheet, loaded after the built-in ones: see the Configuration reference. Its rules reach the inputs and buttons of every dashboard; a sheet that changes the theme's accent colour, for example, changes every slider and switch. A widget's own styles still win over it, unless the sheet marks its rules `!important`.

Sizes of inputs and buttons

- **At least 30 pixels tall.** The theme gives buttons, one-line text boxes, number and date boxes and drop-down lists a minimum height of 30 px. Drawn smaller, the control still renders 30 px high and hangs below its box. `min_height: "0px"` lifts the minimum, and the control then takes exactly the height it was drawn at.
- **The drawn box is the control.** Borders and padding are inside the box, not added to it.
- **Text** is 14 px Open Sans unless `font_size` and `font_family` say otherwise.
- **A slider sizes itself from its box:** the thumb is as thick as the box, at least 10 px, and the track is 8/18 of the thumb, at least 3 px.
- A newly dropped FieldDate is 160 px wide, enough to show a whole date and its picker button.

Dynamic styling

Every key of the record can be a formula of its own, and the record is evaluated again each time the view redraws, so the look follows the plant.

- **A value per state.** `if` chooses: `{fill: if(P101_RUN, "#16A34A", "#9CA3AF")}`. The condition must be a Boolean. A signal held as a number, such as a coil read as 0 or 1, is compared: `if(P101_RUN != 0, ...)`.
- **Thresholds** nest `if`s, the most severe first:

```
{fill: if(to_double(LIT_301) > 4.2, "#DC2626",
       if(to_double(LIT_301) > 3.8, "#F59E0B", "#2563EB")),
 stroke: "#1E3A8A", stroke_width: 1}
```

A comparison also works with an infinite value or NaN, which a division by zero can give: NaN is neither above nor below any threshold, so it takes the last branch.

- **Grey when there is no data.** A variable that was never written, or holds `null`, makes `if` fail; `evl` catches that and gives its second argument. The three-state LED of a pump, with a fourth look for no data:

```
{fill: evl(if(P101_TRIP, "#DC2626", if(P101_RUN, "#16A34A", "#FFFFFF")), "#9CA3AF"),
 stroke: evl(if(P101_TRIP, "#7F1D1D", if(P101_RUN, "#14532D", "#4B5563")), "#6B7280"),
 stroke_width: 3}
```



Running	Stopped	Tripped	No data
P101_RUN true	both false	P101_TRIP true	a signal without a value

- **Smooth changes.** `transition: "fill 0.5s"` makes a fill change colour over half a second instead of at once. Name several properties with commas: `transition: "fill 0.5s, stroke 0.5s"`.
- **Showing and hiding.** Three ways, which differ:
  - `visible`, the widget's **Visible** formula, removes the widget from the view while it is `false`. The editor keeps showing it, so it can still be selected there. This is the way to hide.
  - `display: "none"` in `styles` hides the widget in the editor too, where it becomes hard to find and select.
  - `opacity: 0` makes the widget invisible but leaves it in place: it still takes clicks.

Blinking and rotation

A dashboard has no timer, so no style can switch itself on and off. A blink or a rotation is a CSS animation, named in the `animation` key. The Site provides two animations for dashboards, and keeps their names stable:

Animation	What it does	Style
`do-blink`	Fades the widget from full opacity down to 0.2	`animation: "do-blink 1s infinite alternate"`
`do-spin`	Turns the widget one full turn	`animation: "do-spin 2s linear infinite", transform_box: "fill-box", transform_origin: "center"`

- **The timing is part of the text.** `do-blink 0.5s` blinks twice as fast; `alternate` fades back up instead of jumping; `do-spin 4s` turns half as fast.
- **Switch an animation with the state:** `animation: if(LSHH_301, "do-blink 0.5s infinite alternate", "none")`.
- **An SVG shape turns about its own centre** only with `transform_box: "fill-box"` and `transform_origin: "center"`. Without them it turns about the corner of the canvas.
- **What can move.** The SVG widget and groups take no styles, so they cannot be animated. An Image widget can, for example the picture of a fan or an agitator. RadialLabels and FieldSliderVertical must not spin: the animation replaces their own turn.
- **Only these two names.** The pages of the product hold other animations for their own use; their names may change in any release, so a dashboard must not rely on them.

> **Important**
>
> Both animations stop when the viewer's operating system asks for reduced motion, an accessibility setting; the widget then shows its normal look, and transitions happen at once. A blink must never be the only sign of an alarm: change the colour as well, as the alarm LED does.

The canvas background

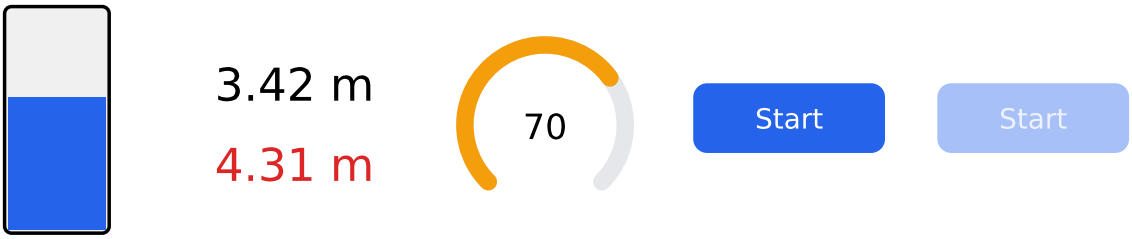
The colour behind the drawing is a setting of the dashboard, not a style: **Background** and **Opacity** in the editor, stored as `#RRGGBB` or `#RRGGBBAA`, white by default.

- **Below 100 % opacity,** a screen shows its own surface through the canvas, which follows the viewer's light or dark theme; a panel shows the dashboard it was opened over.
- **A background that follows the plant** is a widget: a Rect covering the whole canvas, last in the component list, with a `styles` formula. A picture or a gradient behind the drawing is an SVG widget, last in the list.

See The canvas.

Recipes

Each recipe is written for the widget it names, with the variables of a small plant: a tank level `LIT_301`, its high-high switch `LSHH_301`, a pump `P101` and a temperature `TIT_201`.



Level bar
60 %, growing upwards

Value label
red above 4.2 m

Gauge by threshold
amber from 60, red from 80

Rounded button
Action button

Dimmed look
opacity 0.4 when not ready

Alarm LED

A Circle of 24 × 24 pixels for the high-high level switch. Red and blinking in alarm, green otherwise, and grey while the switch has no value:

```
Styles  {fill: evl(if(LSHH_301, "#DC2626", "#16A34A"), "#9CA3AF"),
         stroke: "#1F2937", stroke_width: 2,
         animation: evl(if(LSHH_301, "do-blink 0.5s infinite alternate", "none"), "none")}
```

The colour carries the alarm on its own for a viewer whose system reduces motion.

Level bar that grows upwards

Two Rects over the same box, the inside of a 4.5 m tank. The bar is higher in the component list than the tank, so it is drawn over it, and its offsets take away the empty part at the top, as Offsets explains:

```
Bar, a Rect at x 400, y 100, 60 × 200, above the tank in the list
  Y offset       200 * (1 - clamp(to_double(LIT_301) / 4.5, 0.0, 1.0))
  Height offset  -200 * (1 - clamp(to_double(LIT_301) / 4.5, 0.0, 1.0))
  Styles         {fill: if(to_double(LIT_301) > 4.2, "#DC2626", "#2563EB"), stroke: "none"}

Tank, a Rect at x 400, y 100, 60 × 200, below the bar in the list
  Styles         {fill: "#F3F4F6", stroke: "#374151", stroke_width: 2, rx: 4, ry: 4}
```

A VerticalGauge draws the same bar without offsets: **Value 2** `LIT_301`, **Max value** `4.5`, and the colour in its `valueStyles`. It draws no track of its own, so the tank Rect stays, after the gauge in the list.

Value label with units

A Label, anchored at its right end so that the digits stay in place while the value changes:

```
Label   evl("%.2f m".format(to_double(LIT_301)), "-- m")
Styles  {fill: evl(if(to_double(LIT_301) > 4.2, "#DC2626", "#111827"), "#9CA3AF"),
         font_size: 20, font_weight: 600, text_anchor: "end"}
```

- `%.2f` writes two decimals with the instance's decimal separator: `3.42 m`, or `3,42 m` where the locale uses a decimal comma. That is right for a label to read; only inside a style does a decimal comma break.
- With `text_anchor: "end"`, the Label's anchor point is where the text ends: place it where the units should end.

Gauge colour by threshold

A RadialGauge for `TIT_201`, on its fixed range of 0 to 100 °C. `styles` draws the grey track, and `valueStyles` colours the value arc by threshold:

```
Value 2       TIT_201
Styles        {fill: "none", stroke: "#E5E7EB", stroke_width: 10}
Value styles  {fill: "none",
               stroke: evl(if(TIT_201 > 80, "#DC2626", if(TIT_201 > 60, "#F59E0B", "#16A34A")), "#9CA3AF"),
               stroke_width: 10, stroke_linecap: "round"}
```

Keep `fill: "none"` in both records: a filled arc fills its chord. A Label placed at the gauge's centre shows the number.

Rounded button

For an ActionButton or a FieldButton. It sets its own colours, so it looks the same in the light and the dark theme:

```
Styles  {background_color: "#2563EB", color: "#FFFFFF", border: "none", border_radius: 8,
         font_size: 16, font_weight: 600, padding: "0px 16px", cursor: "pointer",
         box_shadow: "0 1px 3px #0000004D"}
```

- Setting `background_color` removes the theme's hover and pressed tints; the cursor keeps it reading as a button.
- For a button drawn less than 30 pixels high, add `min_height: "0px"`.

Dimmed look when not ready

The same button, dimmed while the pump is not ready to start:

```
Styles  {background_color: "#2563EB", color: "#FFFFFF", border: "none", border_radius: 8,
         opacity: evl(if(P101_READY, 1, 0.4), 0.4),
         cursor: evl(if(P101_READY, "pointer", "not-allowed"), "not-allowed"),
         pointer_events: evl(if(P101_READY, "auto", "none"), "none")}
```

`opacity` and `cursor` change only the look. `pointer_events: "none"` also keeps mouse and touch clicks from reaching the button. A keyboard can still press a button that has the focus, so an Action button's script checks the same condition before it acts.

When a style does not show

What you see	Why	What to do
A fill turns black	The colour is a number from a colour function, a Boolean, or a value the browser does not read	Write colours as text: `"#16A34A"`, or `"#%06X".format(rgb(...))`
Nothing changes at all	The widget takes no styles: the SVG widget, a group, a linear gauge's `styles`. Or the key does not apply to its element, such as `fill` on a button	See SVG widgets and HTML widgets
The look stopped following the plant	The styles formula fails, and the last styles that worked stay	Correct the formula; guard variables that may be missing with `evl`
A decimal has no effect	`format` wrote a decimal comma	Pass the number itself
A size has no effect	A text size without a unit	A number for the length keys, or text with its unit
A font differs on some devices	The family is not installed there	End `font_family` with a generic family
Inputs look different in dark mode	HTML widgets follow the theme	Set `background_color`, `color` and `border_color`
A button no longer changes under the pointer	`background_color` replaces the theme's tints	Expected: add `cursor: "pointer"`
A blink does not blink	The viewer's system reduces motion, or the animation is not `do-blink`	Pair the blink with a colour
A control is taller than drawn	The 30-pixel minimum of the theme	`min_height: "0px"`
A shape turns about the canvas's corner	The turn has no origin of its own	Add `transform_box: "fill-box"` and `transform_origin: "center"`

Next steps

- Properties every widget shares

- How dashboards draw and refresh
- Colour functions
- Gauges

This page describes Data Orchester Dashboards 1.9.4.