Amtiri

REFERENCE MANUAL

How dashboards draw and refresh

Generated on November 25, 2026

A dashboard is a drawing of a fixed size. Its widgets sit at pixel positions on a canvas, and what they show comes from formulas: the text of a label, the colour of a pump, the length of a level bar. The formulas run on the server, in the engine, and the browser receives only what they drew. This page explains the canvas, how a dashboard is scaled to the screen, which widget is drawn over which, and when a dashboard redraws. The widgets themselves are described from Properties every widget shares onwards.

The canvas

The canvas settings sit at the top of the editor's left column, above the component list:

Editor field	Stored as	Default	Meaning
Width, **Height**	`width`, `height`	1024 × 640	The size of the drawing, in canvas pixels. Every widget is placed and sized in these units
Background	`background`	`#FFFFFF`, white	The colour behind the drawing
Opacity	The last two digits of `background`	100 %	How opaque the background is, in whole percent

- **The background is one colour.** It is stored as `#RRGGBB` while it is opaque, and as `#RRGGBBAA` below 100 %. It is a setting, not a formula.
- **Below 100 %, what lies behind the canvas shows through.** On a screen, that is the view's own surface, which follows the viewer's light or dark theme. In a panel, it is the dashboard the panel was opened over.
- **A background that changes with the plant, or that carries a picture,** is a widget: a Rect or an SVG widget that covers the whole canvas, placed last in the component list. See the canvas background.

Fit and zoom

A dashboard opens at 100 %: one canvas pixel is one pixel of the page, and a canvas larger than the view scrolls. The rail at the right of the view scales it:

Button	What it does
Zoom in	Enlarges the drawing by 10 % of its current size
Actual size	Returns to 100 %
Zoom out	Shrinks it by about 5.7 %. The step is deliberately not the reverse of Zoom in, so that a few clicks in each direction reach any size
Fit to view	Scales the whole canvas to fit the view
Full screen	Shows the view on the whole screen and fits it. Leaving full screen fits it again

- **Fitting never distorts.** The canvas is scaled by the smaller of two ratios: the view's width over the canvas width, and the view's height over the canvas height. The whole drawing is on screen, and it fills the view only when both have the same proportions; otherwise an empty band remains at the right or at the bottom.
- **Everything scales together:** the drawing, the inputs and buttons over it, and the panels opened on it.
- **The zoom belongs to the open view.** It is not saved, and every dashboard opens at 100 %.
- **On a phone,** a dashboard is scaled to the width of the screen, never above 100 %, and scaled again when the phone turns. The phone lists dashboards only, never panels.
- **The editor has its own zoom,** from 25 % to 300 %. It never changes the canvas size.

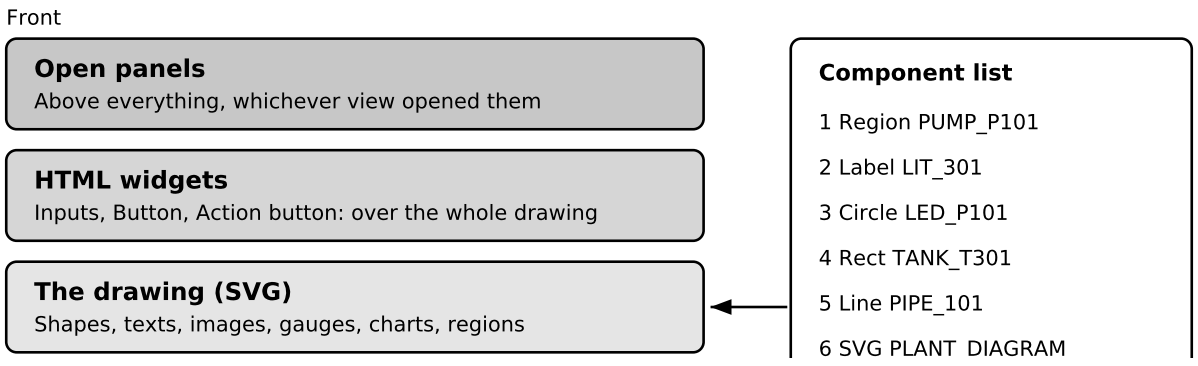
> **Tip**
>
> Give the canvas the proportions of the area that will show it. A dashboard shown in full screen has the whole screen except the zoom rail at its right; one shown in a tab has less. A canvas with the same proportions as that area fills it when fitted, with no empty band.

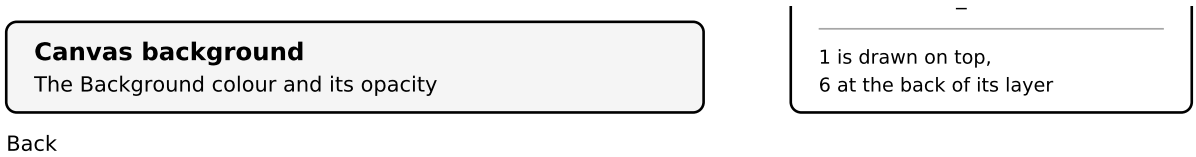
Paint order and layers

The component list in the editor is the paint order: **the first widget in the list is drawn on top**, and the last one at the back.

- A widget dropped from the palette goes to the top of the list, so it is drawn over everything.
- A pasted widget goes just above the widget that was selected, or to the top when nothing was.
- **Move up** and **Move down**, the arrow buttons above the list, move the selection one place.
- Inside a group the same rule holds among the group's members: the first member is on top.
- A drawing that fills the whole canvas, such as a plant diagram in an SVG widget, therefore goes **last**. Everything else is drawn over it.

The order decides only within a layer. A view is made of four layers, and no widget leaves its own:





- **Inputs and buttons are always in front of the drawing,** whatever their place in the list: a Rect first in the list is still drawn under a Number field. Among themselves, the first in the list is on top.
- **A Region can be clicked only where nothing is drawn over it.** Keep it above the shapes it covers, that is, higher in the list; a shape or a label above it takes the click. A region can never cover an input or a button. See Region.
- **A panel opened by a click closes at the next click outside it,** and that click does not reach the widget under it.

Refresh

A dashboard has no timer. It redraws when a variable it shows changes:

1. When a view opens, it collects the variables named in every formula of every widget, and the variable of every input. These are the view's subscriptions.
2. When the engine changes one of those variables, the whole view is evaluated again: every formula of every widget, not only the formulas that name the variable.
3. The browser then receives what changed in the drawing.

- **Only a change counts.** A variable written again with the value it already holds changes nothing, and no view redraws. A formula variable redraws the views that show it only when its result changes.
- **Changes are handled one at a time,** in the order they came, for each viewer. A variable that changes ten times a second redraws every view that shows it ten times a second.
- **A name in quotes is text, not a subscription.** `history(PLANT_LOG, "LIT_301", addHours(now(), -1), now())` does not redraw when `LIT_301` changes.
- **The subscriptions are fixed when the view opens.** After changing a dashboard, open it again.

A clock on a dashboard

`now()` never redraws anything by itself: a view redraws only when a variable changes. A clock therefore reads a variable that changes, recalculated by a Scheduler:

Where	Setting	Value
Variable `CLOCK`	Formula	`now()`
Scheduler, one definition	Variable and CRON	`CLOCK` , `* * * * * ?` (every second)
Label	Label	`"%tT".format(toLocale(CLOCK))`

The label shows the time as `14:05:09`, in the viewer's time zone. A CRON of `0 * * * * ?` updates the clock once a minute, which is enough for a label written as `"%tR".format(toLocale(CLOCK))` (`14:05`) and redraws the view sixty times less often.

When a formula fails

A formula that runs but fails, because it reads a variable that was never written or gets a value of the wrong type, falls back silently. Nothing is logged, and the rest of the view draws as usual:

- `visible`: the widget is shown.
- The offsets (`dX` , `dY` and the others): 0.
- The gauges' value formulas: the fixed value stored with the widget.

- `styles`: the last styles that worked; none, if it never worked.
- A chart's `min` and `max`: 0 and 100.
- A label: blank.

Guard a formula that reads a variable which may never have been written with `evl`, which gives its second argument when the first fails: `evl("%.2f m".format(to_double(LIT_301)), "-- m")`.

A formula that does not compile behaves differently. In the viewer, its widget is no longer updated and the error is written to the server log. In the editor, a message titled `Component:` and the widget's name appears each time the drawing is redrawn. The formula editor already shows the error while the formula is written: correct it before applying.

Editor and viewer

The editor runs the formulas too, so a dashboard shows plant values while it is built. It evaluates them each time it redraws the drawing, which is after every edit, with the values of that moment; it does not follow the plant between edits.

- **Evaluated in the editor and in the viewer:** `styles`, a label's text, a button's label, the lists of labels, values and options, a chart's `min` and `max`, `valueStyles`, and the gauges' value formulas.
- **Evaluated only in the viewer:** `visible`, the offsets `dX`, `dY`, `dWidth`, `dHeight`, `dToX` and `dToY`, and a Region's **Popup dX** and **Popup dY**. The editor draws every widget where it was placed, so that it can be found, selected and moved: a widget hidden by `visible` stays visible in the editor.
- `display:"none"` in `styles` hides a widget in the editor too, which makes it hard to select. Prefer `visible`.
- Inputs and buttons cannot be operated in the editor, a switch shows no state there, and an Action button's script runs only when the button is clicked in a view.

Time zone

The engine keeps dates and times in UTC: `now()` gives the current UTC date and time, and a logger stamps its rows in UTC. `toLocale(...)` converts a UTC date and time into the configured time zone, and **while a view redraws, that zone is the viewer's** : the offset the viewer's browser reports. So `toLocale(CLOCK)` shows each viewer their own local time, and two viewers in different zones read different clocks. The editor converts with the instance's own time zone. The types chapter describes dates and zones in full.

- **The offset is taken to the minute,** so a viewer in a zone with a half-hour offset, such as UTC+05:30, sees their own time. Until the browser has reported its offset, the view uses the instance's time zone.
- **Keep variables in UTC.** Use `toLocale` in widgets only, at the point where a date is shown.

> **Caution**
>
> The zone is switched for the whole engine while a view redraws. A variable's formula that the engine recalculates at that same moment, and that calls `toLocale`, can see the viewer's zone instead of the instance's. Keep `toLocale` out of the formulas of variables.

Widget types a release does not know

A dashboard can hold a widget type that the running release does not know: one saved by a newer release, or one of the names older releases used (`Group`, `HGrid`, `VGrid`, and `ChartPie`, a pie chart that was never offered). Such a widget is skipped and a warning is written to the server log; the rest of the dashboard opens and draws as usual.

The editor does not show such a widget, but it keeps it: saving the dashboard writes it back unchanged, in its place in the list, inside its group if it had one. Edit a dashboard with the release that wrote it, or a newer one, to see and change every widget. Moving dashboards between instances, give the receiving instance the same or a newer release.

Performance

- **Every change evaluates the whole view.** The work grows with the number of widgets times the number of changes a second. Keep the number of widgets and distinct variables on one dashboard modest, and move detail into panels, which cost nothing until they are opened.
- **Calm a fast variable before showing it.** A formula variable such as `FIT_101_SHOWN`, with the formula `round(FIT_101, 0.1)`, changes, and redraws the views that show it, only when the rounded value changes. Show it instead of `FIT_101`.
- **Prefer one SVG widget to large Image widgets.** An Image widget encodes its picture again at every redraw; an SVG widget does not.
- **Keep a RadialGauge's `valueStyles` short.** It is compiled again at every redraw.
- **Every open view subscribes on its own.** Each tab and each open panel redraws with its own variables; closing it ends its subscriptions.
- Opening a dashboard shows the wait indicator until the view is built.

Who sees a dashboard

- **The Role** in a dashboard's settings decides who is offered it. Empty, the dashboard is listed for everyone; with a role, it is listed in the menu and on the phone only for the users who hold that role.
- **Creating and saving dashboards** needs the `CONFIGURATION` role.
- **Displaying** dashboards and panels needs an edition that includes them. Editing never does: an instance whose licence does not show dashboards can still build and keep them. See How licensing works.
- **A dashboard's formulas run on the server** and can read every variable of the instance, whoever looks at it. The role decides who is offered a dashboard, not what its formulas can read.

Panels

A panel is a dashboard whose **Category** is **Panel**. It declares **Parameters**, names that its widgets use in place of variables, and it is opened over a view, bound to that view's variables: by a Region, or by `openPanel` in an Action button's script. It is closed by the rules of whatever opened it, or by `closePanel`.

- A panel is drawn in the top layer, at the zoom of the view it was opened on, with the size of its own canvas.
- Its background can be transparent: the dashboard below then shows through. The popup has rounded corners and a shadow, and fades in and out.
- Panels are never listed in the menu or on the phone, and cannot be a user's default dashboard.

Regions and panels describes parameters, binding, placement and closing. `openDashboard` replaces the content of a view with another dashboard.

Next steps

- Properties every widget shares
- Styling widgets
- Regions and panels

- Scheduler

This page describes Data Orchester Dashboards 1.9.4.