Heatmap Widget for Tridium Niagara

User Guide

baudrate.io

niagara4

September 12, 2026

Contents

1 Introduction **2**

2 Requirements **2**

3 Quick Start **3**

4 Floor Plan **5**

5 Coordinates **5**

6 Config Editor **6**

7 Configuration **8**

8 Sensors **8**

9 Zones **9**

10 Colormaps **12**

11 FAQ **13**

1 Introduction

The Heatmap Widget renders a continuous, interpolated map of any scalar field over a customer-supplied floor plan from sparse sensor readings. It is designed for data centers, pharma cleanrooms, cold storage, multi-zone HVAC, warehouses, and any other application that benefits from spatial visualization of temperature, humidity, CO_2, pressure, or any other numeric quantity.

The widget runs in the Niagara HTML5 viewer (and in Workbench via **Browser Preview**) using standard Niagara point subscriptions, and redraws as values change.

Key features:

- Smooth thermal field rendered live over the floor plan, exact at every sensor and blended into a continuous gradient between sensors.
- Rendered in the browser with WebGL – no extra server, no JACE load.
- Drop-in widget that overlays the floor plan in any Niagara graphic, with any number of sensors per widget.
- Wall-aware interpolation: internal walls and partitions keep the interpolated field within physical boundaries, so the rendered field does not bleed across racks, cleanroom partitions, or any other geometry the integrator defines as a wall.
- Multi-zone layout with per-zone colors, scale, opacity, and rendering tuning. A cold aisle can use a blue 16-22 °C band while an adjacent hot aisle uses a red 28-40 °C band, both visible at meaningful resolution on the same canvas. Because each zone can have its own scale, the same color can represent different values in different areas of the screen – a deep red in the cold aisle means 22 °C, while a deep red in the hot aisle means 40 °C.
- Various colormap presets, plus a custom gradient editor.
- Adjustable overlay opacity so the floor plan stays visible underneath.
- Tunable smoothness – softer for broad trends, sharper for hot-spot definition.
- Live updates from standard Niagara points.

2 Requirements

- Niagara-powered device with software N4.8 or later, such as JACE, Supervisor, or any OEM version
- Heatmap widget module and license
- Floor plan image of the area to visualize

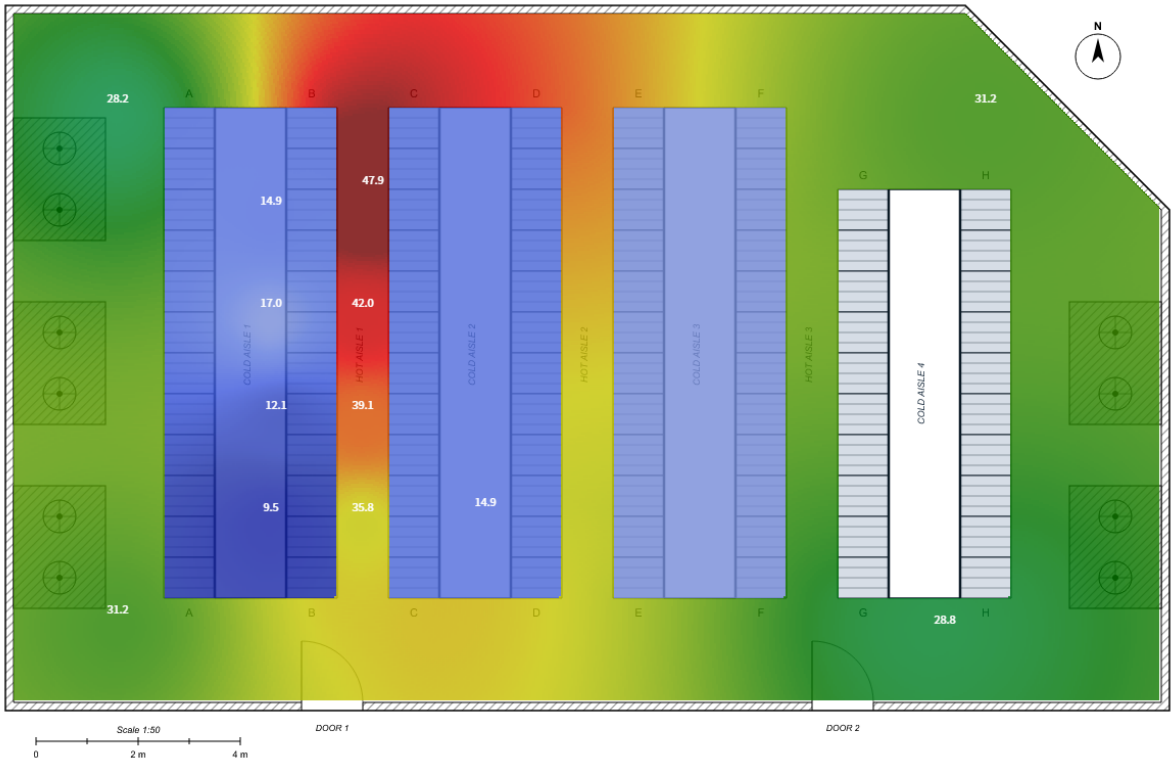

Figure 1: Heatmap widget rendered over a data center floor plan

3 Quick Start

1. Copy `heatmap-ux.jar` and `bdrtLicense-rt.jar` to the Niagara `/modules` folder and restart the Workbench software.

   a. If the widget is hosted on PC, restart the Supervisor station.
   b. If the widget is hosted on JACE, first use the **Software Manager** view on the JACE platform to import both jars with all their dependencies, then restart the JACE station.

2. Install the license. Open the **heatmap** palette and add **LicenseService** to the station's **Services**. Open the new **Services > LicenseService > Offline**. Press the import icon » next to the **License** property and import the supplied `.license` file. Press **Save** and verify **Status Message** shows *Valid*, as shown below.

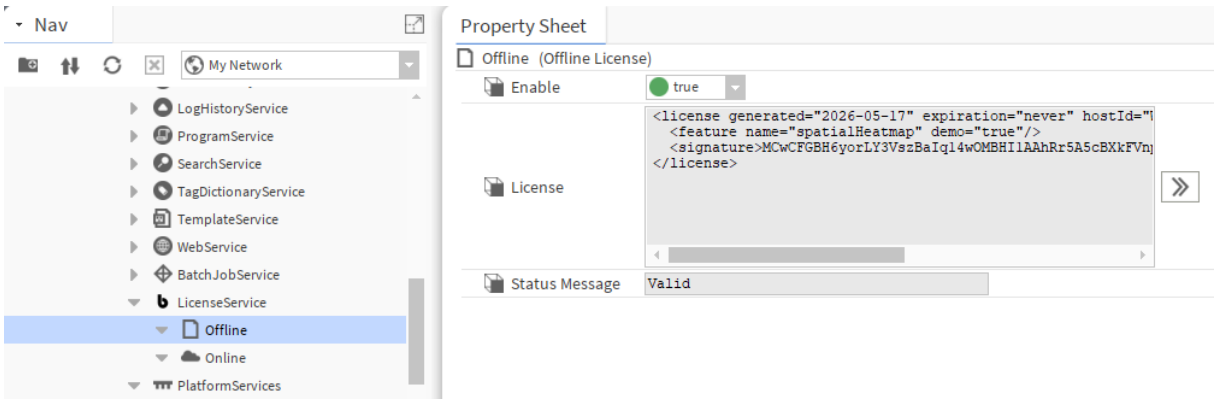

Figure 2: Offline license installed and validated

3. Try the example. Drag the **Example** folder from the **heatmap** palette into the station. Double-click the example component and click the **Browser Preview** button – the example heatmap renders as shown at the top of this guide. This confirms the module is installed correctly and the license is active.

To build your own heatmap, assuming the example is working, follow these steps:

1. Add the floor plan to the Px page by dragging the image file onto the page – Workbench creates a **Label** widget that displays the image. For the alternative transparent-overlay layering, see **Floor Plan**.
2. Add the Heatmap Widget. Drag **Heatmap Widget** from the **heatmap** palette on top of the floor plan, then position and resize it so it covers the area you want to visualize.
3. Open the widget's **config** (see **Config Editor**) and set **offset** to the widget's top-left position on the Px page (copy `x`, `y` from the widget's layout slot; use `0`, `0` if it sits at the page origin). See **Coordinates** for why the offset is needed.
4. Set **zones[0]**'s **radius** to `10`. This makes each sensor render as a small bright dot, so each sensor's exact position is immediately visible against the floor plan – the alignment aid for the drag-and-drop placement in the next step. This is a temporary setup value, not the final operating one.
5. Add each sensor: **Add Child** to **sensors**, set **pos**, and **Animate value** to the bound Niagara point – see **Sensors**.
6. Reset **zones[0]**'s **radius** to `0` (or to the desired operating value). With sensors in place, the alignment aid is no longer needed; resetting **radius** makes the heatmap render as a continuous field instead of isolated dots, which is the normal operating mode.
7. Add sub-zones for any contained area that needs its own colors or scale (cold aisle, cleanroom, freezer) using the **Shapes** palette helpers – see **Drawing Walls with Shape Helpers**.
8. Set **colors**, **scale**, **opacity**, **power**, and **radius** – on **zones[0]** for the default appearance, and

on individual sub-zones only where they need to differ from the main zone (typically a different scale band). See **Zones** for the property reference.

Save the Px page and switch to **Browser Preview** mode. The heatmap renders as soon as the first bound value arrives.

4 Floor Plan

The floor plan is not part of the Heatmap Widget. The widget renders only the interpolated heatmap on a transparent canvas; the floor plan is a separate image on the Px page that the integrator layers either underneath or on top of the widget.

There are two layering options, both standard Px arrangements:

- **Floor plan underneath the widget** (typical case). The floor plan is opaque and shows the room, racks, walls, equipment, and labels. Drag the image file onto the Px page – Workbench creates a **Label** widget that displays it – then drop the Heatmap Widget on top of it, sized to cover the area being visualized. The widget's **opacity** controls how much of the floor plan shows through.
- **Floor plan on top of the widget** (transparent overlay case). The floor plan image has a transparent background and contains only walls, labels, equipment outlines, or other annotations. Place the Heatmap Widget first, then drop the floor plan image on top of it. The heatmap shows through the transparent parts of the floor plan, while the opaque annotations stay crisp on top of the heatmap. This is useful when annotations must remain readable regardless of the heatmap colors underneath.

In both cases, the Heatmap Widget itself is unaware of the floor plan – it only knows about its own sensors and zones.

5 Coordinates

Sensor positions (**sensors[N].pos**) and zone walls (**zones[N].walls**) are specified as **page-absolute pixels** – the same coordinate space that standard Niagara widgets (labels, lines, rectangles, polygons) use in their `layout` slot. The widget then subtracts its own **offset** slot before rendering, so the painted pixel lands at the correct spot on the canvas.

This is a deliberate workflow choice. Every Niagara label or shape widget records its on-page position as an `x,y,w,h` string (for rectangular widgets) or a point list (for polygons) in its `layout` slot. By accepting page-absolute pixels in **pos** and **walls** and translating internally via **offset**, the Heatmap Widget lets the integrator copy a label's or shape's **layout** value verbatim into a sensor's **pos** or a zone's

walls – no per-coordinate arithmetic, no manual offset subtraction. This is what the recommended sensor and wall workflows rely on:

- Sensors are placed by dropping the bound Niagara point onto the Px page as a label, then copying the first two numbers from the label's **layout** into the sensor's **pos** – see **Sensors**.
- Walls are drawn by dropping a **Line**, **Rect**, or **Polygon** helper from the **Shapes** folder onto the Px page, then copying its **layout** into the zone's **walls** – see **Drawing Walls with Shape Helpers**.

To configure **offset**:

- Set **offset** to the widget's own top-left position on the Px page – the first two numbers from the widget's own `layout` slot. Use `0,0` if the widget sits at the page origin.
- If the widget is later moved on the page, update **offset** to match. The **pos** and **walls** values do not need to change.

6 Config Editor

The widget's **config** property is a tree of structured attributes – top-level slots (**debug**, **offset**, **sensors**, **zones**), array entries (one per sensor or zone), and primitive values (positions, ranges, opacities, colors, and so on). It is edited through the config editor, a dedicated modal dialog that opens when the **config** property is clicked. The dialog has three regions:

- **Attribute tree (top)** – the tree of attributes with three columns: **Name**, **Value**, and **Ord** (which shows the bound ord for any animated attribute). Click the + and − icons to expand or collapse a branch; click a row to select an attribute.
- **Button bar (middle)** – the actions that operate on the selected attribute.
- **Help text (bottom)** – the schema description of the selected attribute: its name, type, default, allowed range, and meaning.

Buttons:

- **Add Root** – add a top-level attribute. Not used for the heatmap config: the four root slots (**debug**, **offset**, **sensors**, **zones**) are fixed by the schema.
- **Add Child** – add a child entry under the selected attribute. Used on **sensors** and **zones** to append a new sensor or zone, populated with the schema's default values.
- **Remove** – delete the selected attribute or array entry.
- **Edit** – edit the selected attribute's value. Enabled only for primitive types (string, number, boolean); disabled for objects and arrays. The **colors** attribute opens a gradient picker.

- **Animate** – bind the selected value attribute to a Niagara point via the standard ord chooser. Used on a sensor's **value** to track a live numeric point. The bound ord then appears in the **Ord** column; press **Animate** again to un-bind.
- **Expand / Collapse** – expand or collapse every root in the tree.

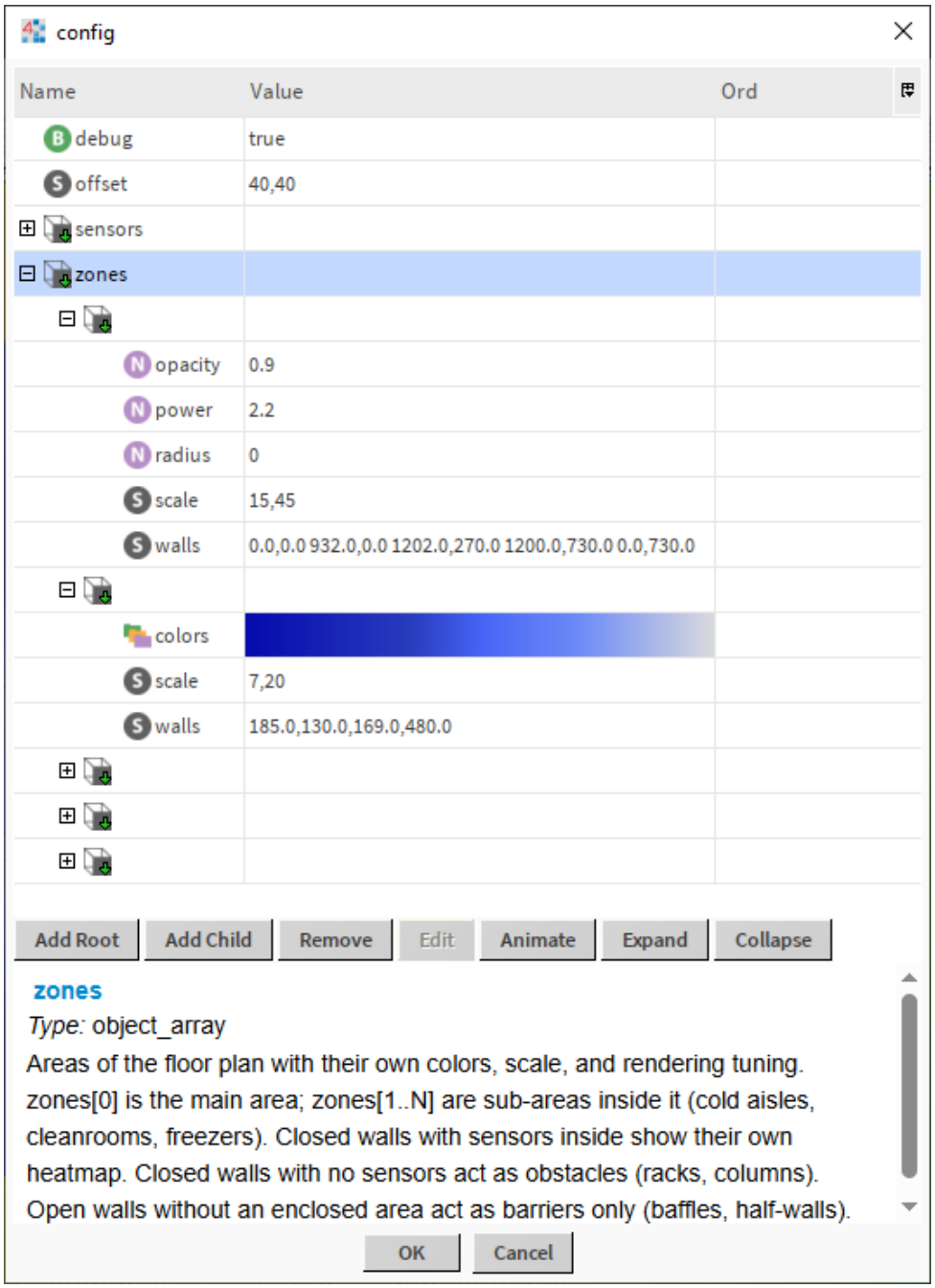

Figure 3: Config editor showing the attribute tree, button bar, and help text

Press **OK** to apply changes; press **Cancel** to discard.

Defaults are sensible – you do not need to set every attribute. Add and modify only the attributes that need to differ from the default.

7 Configuration

The widget's **config** holds the entire widget configuration. Top-level properties:

- **debug** – when enabled, the widget prints extra information to the browser console. Enable only when asked by Baudrate support to capture diagnostic output.
- **offset** – the widget's top-left position on the Px page as `x,y`. Copy this from the widget's layout slot. Use `0,0` if the widget is at the page origin. The widget uses this to translate page-absolute sensor and wall coordinates into widget-local coordinates, so the same coordinate values keep working if the widget is later moved on the page (provided **offset** is updated to match).
- **neighbors** – how many nearby sensors are blended at each point on the plan. `8` (the default) suits almost every plan and keeps the calculation time independent of how many sensors the widget has. Raise it towards `16` for zones with a low **power**, where distant sensors still carry weight. `0` blends every sensor, which is the exact calculation but slows down noticeably past about 40 sensors, since each sensor then needs its own distance map.
- **gridPixels** – the size of one calculation cell in floor plan pixels. `0` (the default) chooses a size automatically from the plan size and the **neighbors** setting. `1` is the sharpest and the slowest to open; higher values open faster but make the coloured area around each sensor more angular. Raise it if a large plan is slow to open, lower it if the result looks blocky. It affects only how long the view takes to open, not how smoothly it updates afterwards.
- **sensors** – the list of sensor points placed on the floor plan. See **Sensors**.
- **zones** – the list of areas with their own colors, scale, and rendering tuning. **zones[0]** is the main area; **zones[1..N]** are sub-areas inside it (cold aisles, cleanrooms, freezers). See **Zones**.

8 Sensors

A sensor is a point on the floor plan with a live numeric value. The widget reads the value via standard Niagara point subscription and uses all sensor values to render the heatmap.

The widget is unit-agnostic – it interpolates whatever raw numbers the bound points report. The **scale** slot on each zone must be in the same units as the bound points (°C, °F, %RH, ppm, Pa, dBm, occupancy counts, and so on).

Sensor properties:

- **pos** – position on the floor plan as `x , y` in pixels (page-absolute). The widget subtracts the **offset** slot from this value to translate into widget-local coordinates.
- **value** – current reading. Right-click and select **Animate** to bind to a Niagara numeric point.

The recommended workflow places every sensor visually on the Px page using standard Niagara drag-and-drop, then copies the resulting pixel coordinates into the widget configuration. This avoids measuring pixel positions by hand and keeps the sensor entries aligned with what the operator sees on the floor plan.

Before placing the first sensor, set **zones[0]**'s **radius** to a small value (for example, `10`). This limits each sensor's contribution to a tight bright dot of about that many pixels, so each sensor's exact position is immediately visible against the floor plan once a value is bound.

For each sensor:

1. **Drag the Niagara numeric point onto the Px page.** Workbench creates a label widget that displays the live value.
2. **Position the label** exactly where the sensor sits on the floor plan. The label can be left in place as a live readout next to the heatmap, or hidden later if only the heatmap is needed.
3. **Copy the label's position.** Open the label's **layout** slot. The value is `x , y , w , h`, where `x , y` is the top-left corner of the label (not its centre); copy the first two numbers. Position the label so its top-left corner sits where the sensor is on the floor plan, or compensate by adjusting **pos** later in step 7.
4. **Add a sensor entry.** In the Heatmap Widget's **config**, right-click **sensors** and select **Add Child**. A new entry appears with default **pos** = `0 , 0` and **value** = `0`.
5. **Paste the coordinates into pos.** The pasted value goes straight in – **pos** takes only `x , y`.
6. **Bind value to the same point.** Right-click the new sensor's **value** and select **Animate**; pick the same Niagara point that drives the label.
7. **Verify alignment.** Save the page. The widget repaints with a small bright dot at the new sensor's position; the dot must sit directly under the label. If it does not, adjust **pos** by a few pixels or check the widget's **offset**.

After the last sensor is in place and verified, reset **zones[0]**'s **radius** so the heatmap renders as a continuous field instead of isolated dots.

To remove a sensor, right-click the sensor entry and select **Remove**.

9 Zones

A zone is defined by one or more walls. When the walls enclose an area, the zone is that area – with its own color gradient, value scale, opacity, and rendering tuning. When the walls are a single line,

the zone has no area of its own; it acts only as a barrier that stops the surrounding heatmap from propagating across it.

Zones primarily model physical walls – containment glass, partitions, rack rows, cleanroom boundaries. Heat does not propagate across a zone boundary, so a cold-aisle sensor never bleeds into the adjacent hot aisle, and a freezer never averages with the anteroom outside it. This is the main reason zones exist, and it applies even with a single colormap shared across the floor plan.

Per-zone scale, colors, and opacity then layer on top of that, enabling correct visualization of mixed-band layouts – for example a cold aisle (16-22 °C, blue band) next to a hot aisle (28-40 °C, red band) rendered at meaningful resolution on the same canvas.

Zone properties:

- **walls** – walls that define the zone shape, in page-absolute pixel coordinates. Use `x,y,w,h` for a rectangle, `x1,y1 x2,y2 ...` for a custom shape (polygons close automatically). Separate multiple walls with `;`. Empty for **zones[0]** means the heatmap fills the full rectangular widget area. Set **zones[0]**'s **walls** to a closed polygon when the heated area is not rectangular (an L-shaped room, a building outline, an irregular hall) – the heatmap is then clipped to that polygon and everything outside it renders transparent, so the widget no longer paints over walls, corridors, or other parts of the floor plan that are not part of the area being visualized.
- **visible** – draw this zone's walls on the floor plan as grey lines. Off by default, so walls stay hidden. Independent of **enabled**, so you can show walls that are not used in the calculation, or hide walls that are used. Sub-zones use the main zone's value if left at the default.
- **enabled** – use this zone's walls as thermal barriers. On by default. Turn it off to treat the zone as if its walls are not there: the zone stops being a separate area, with no barrier and no distinct region, so the surrounding heatmap flows straight through it. Use it to compare against plain distance weighting that blends across the walls. Sub-zones use the main zone's value if left at the default.
- **colors** – the color gradient for this zone. Double-click the swatch to pick a preset or build a custom gradient. See **Colormaps**. Sub-zones use the main zone's colors if left at the default.
- **scale** – the value range as `min,max`, for example `16,22` for a cold aisle. Values below min show as the lowest color; values above max show as the highest. Sub-zones use the main zone's scale if left at the default.
- **opacity** – transparency for this zone. `1` is solid, `0` is invisible. Lower values let the floor plan show through more. Default `0.8`. Sub-zones use the main zone's opacity if left at the default.
- **power** – controls how sharp or smooth the gradient looks. Higher values (3-6) give sharper hot or cold spots; lower values (0.5-1.5) give smoother blends. Default `2.0` works for most layouts. Allowed range: `0.5` to `6`.
- **radius** – distance in pixels beyond which the heatmap fades to transparent. Use a smaller value

to hide areas far from any sensor. `0` means no fade. Sub-zones use the main zone's radius if left at the default. Allowed range: `0` to `2000`.

9.1 Wall Geometry

Walls define the zone shape and act as barriers. A sensor on one side of a wall does not influence pixels on the other side. This produces a physically correct visualization in environments with containment (cold aisle containment, cleanroom partitions, cold storage anterooms).

Wall syntax supports three shapes:

- **Rectangle** – `x,y,w,h` defines a rectangle with the top-left corner at `(x, y)` and the given width and height. Example: `100,50,400,200` is a 400-by-200 rectangle starting at pixel `(100, 50)`.
- **Line** – `x1,y1 x2,y2` defines a single wall segment. A line zone acts as a barrier only – it stops the surrounding heatmap from propagating across it but has no enclosed area of its own.
- **Polygon** – `x1,y1 x2,y2 x3,y3 ...` (three or more points) defines a closed polygon. The widget closes it automatically; the first point does not need to be repeated at the end. Polygons enclose an area and can hold sub-zones with their own heatmap.
- **Multiple walls** – separate independent walls with `;`. Example: `100,50,400,200 ; 600,50,400,200` defines two adjacent rectangles in the same zone.

Wall coordinates are page-absolute pixels and are translated by the widget's **offset** slot the same way sensor coordinates are.

9.2 Drawing Walls with Shape Helpers

The **heatmap** palette includes a **Shapes** folder with three helper widgets that make wall drawing visual instead of numeric. Each one uses standard Niagara drag, resize, and move handles, so a wall is drawn the same way any Px widget is positioned. Once the shape sits where you want the wall, its **layout** slot already contains the exact wall string the widget expects – copy it and paste it into a zone's **walls** slot.

The three helpers map to the three wall syntaxes:

- **Line** – a single segment. Use for barriers that do not enclose anything: baffles, half-walls, air-flow dividers.
- **Rect** – a rectangle. Use for rectangular containment areas (cold aisle, freezer, cleanroom bay) and for rectangular obstacles (racks, columns, equipment cabinets).

- **Polygon** – a multi-point closed shape. Use for non-rectangular containment areas (an irregular cleanroom, a curved cold aisle) and for non-rectangular obstacles.

To draw and apply a wall:

1. **Draw it on the Px page.** Drag the appropriate shape from the **Shapes** folder onto the Px page. Move, resize, or edit its points so it overlays the wall, area, or obstacle exactly where you want it on the floor plan.
2. **Copy its layout.** Open the shape's **layout** slot and copy the value. The layout string is already in the wall syntax the widget expects – `x,y,w,h` for **Rect**, `x1,y1 x2,y2` for **Line**, and a point list for **Polygon**.
3. **Paste into a zone.** Right-click **zones** and select **Add Child** to create a new zone. On the new zone, right-click and select **Add Child** to add a **walls** attribute, then paste the layout into its value. Configure the new zone's **colors**, **scale**, **opacity**, **power**, and **radius** as needed, or leave them at the defaults to inherit from **zones[0]**.
4. **Remove or keep the helper.** The helper shape itself can be left visible on the Px page – it doubles as a rendered floor-plan outline drawn over the heatmap – or it can be removed once its layout is copied. The widget renders walls from the **walls** strings only.

10 Colormaps

The widget ships with 22 presets (sorted alphabetically in the picker):

Blackbody, Bluered, Blues, Cividis, Earth, Electric, Greens, Greys, Hot, Jet, JetDark, Picnic, Portland, Rainbow, RdBu, RdYlBu, RdYlGn, Reds, Spectral, Viridis, YlGnBu, YlOrRd.

Default for new zones is **JetDark**, a darker, more saturated variant of the classic Jet that reads well on most floor-plan backgrounds.

Scale mapping. The colormap's `0%–100%` range maps to the zone's **scale** `min–max`. For example, with **scale** `16,22` the `0%` color is `16 \textdegree{}C` and the `100%` color is `22 \textdegree{}C`. Values outside the range clamp to the nearest end.

Per-zone choice. Each zone has its own **colors** slot, so a cold aisle can use `Blues` while an adjacent hot aisle uses `Reds`. Sub-zones that leave **colors** at the default inherit the main zone's gradient.

Custom gradients. Double-click a zone's **colors** field to open the gradient editor. The dropdown at the top selects a preset; the editor below adjusts the color stops directly. Custom edits are saved back to the slot as a list of color stops and are independent of the preset list.

Accessibility. For operators with red-green color vision deficiency, prefer the perceptually uniform, colorblind-safe presets **Viridis** and **Cividis** over **Jet**, **JetDark**, or **Rainbow**.

11 FAQ

11.1 I cannot edit the config in the example view

The example Px view is stored inside the **heatmap** jar and is read-only. To get an editable copy, create a new view and copy the content into it. Right-click the **Example** folder, select **Views > New View**, choose the **Px View** type, and give it a name. Open the read-only example view in the Px Editor, select all of its content and copy it, then open the new view in the Px Editor and paste. The new view is editable and the widget configuration can be changed from there.

11.2 Why does the heatmap not bleed across rack rows?

The widget honors wall geometry. A sensor on one side of a wall does not contribute to pixels on the other side, because the path between them goes around the wall and is much longer than the straight line.

11.3 How do I show racks without giving them their own heatmap?

Define each rack as a closed rectangle or polygon in **zones[0]**'s **walls** slot, with no sensors inside it. The walls block interpolation through the rack and the rack area renders as transparent.

11.4 Why does the heatmap only update every two seconds?

Niagara batches value updates from the station to the browser with a default 2-second coalesce window. The widget itself redraws on every change; the cadence is determined by the station-side delivery rate, not by the widget.

11.5 Why does the widget show with a white background in Workbench?

Workbench's embedded browser does not enable transparent rendering, so any transparent area of the widget renders as white inside Workbench. This is a Niagara platform behavior. The widget renders correctly with transparency in the HTML5 viewer and in any browser.

For Workbench-side preview, click the **Browser Preview** button. Browser Preview uses the full HTML5 viewer pipeline and renders with correct transparency.

11.6 The colormap looks washed out in one zone

Check the zone's **scale** setting. If the value range is much larger than the actual sensor readings, most sensor values map to the middle of the gradient and the colors look muted. Tighten **scale** to the actual range of readings, for example `16,22` for a cold aisle that runs 18-20 °C in practice. The widget then uses the full color range.

11.7 Sensors are missing or in the wrong place

Verify that the sensor's **pos** is correct. The widget interprets **pos** as page-absolute pixels and subtracts the widget's **offset** – a wrong **offset** moves every sensor by the same amount. Also verify that the binding to the Niagara point is active: right-click **value** and check that the binding is not broken.

11.8 A sensor in one zone is showing up in another zone

The widget filters sensor contributions by reachability. If a sensor in zone A reaches pixels in zone B (because the walls do not fully separate them), it will influence both. Check that the wall geometry fully encloses the sub-zone and that no gap exists between adjacent walls.

11.9 Can the widget visualize non-temperature values?

Yes. The widget interpolates any numeric value; the color scale is configured per zone via **scale** and **colors**. Use it for humidity (%RH), CO_2 (ppm), differential pressure (Pa), occupancy counts, signal strength (dBm), or any other scalar field on a floor plan.

11.10 How many sensors does one widget support?

Performance is comfortable up to several hundred sensors per widget, because with the default **neighbors** setting the calculation cost depends on the size of the floor plan rather than on the sensor count. Setting **neighbors** to `0` (blend every sensor) reverses that: the calculation then grows with the number of sensors and becomes slow to open past about 40. Beyond that, consider splitting the layout into multiple widgets (one per floor, per logical zone, or per measurement type), which also keeps the update cadence per widget snappier.