Ignition 8.3 User Manual — Perspective Component Reference

Generated 2026-10-12 from https://www.docs.inductiveautomation.com/docs/8.3/

Appendix

The appendix is your quick reference to all the Components, Expression Functions, and Scripting Functions in Ignition. Each item has a definition, property references, and examples of how to use it.

- Components
- Expression Functions
- System Functions
- Reference Pages

Components

Components are what give you flexibility in designing HMI and SCADA that reflect your company's design and your site's layout. Components are the widgets you deal with every day: buttons, text areas, dropdowns, charts, gauges, linear displays, and so on. The Perspective and Vision Modules come with a host of useful components out of the box, many of which are specialized for industrial controls use. Other modules, like the Reporting module, add more components for specialty purposes.

Configuring components is usually the bulk of a designer's work when designing a project. The basic workflow is to take a component from the palette and drop it into a container or a window. From there, you can use the mouse to drag and resize the component into the correct position. While the component is selected, you can use the component's properties to alter the component's appearance and behavior.

In this section, you can view all the components in the following modules, their properties along with an example of how to configure it.

📂 **Perspective Components**

📂 **Vision Components**

📂 **Report Design Components**

Perspective Components



This section covers all the built-in Perspective components. While a component is selected, you can use the Property Editor to alter the component's properties, which changes the component's appearance and behavior. Components are the building blocks of the Designer Interface that when combined create the visual part of a view to do something useful, like display dynamic information or control a device.

Here is a complete list of Perspective components and a link pointing to a page containing the component's description, properties, and usage examples.

- **Chart Palette**
- **Container Palette**
- **Display Palette**
- **Embedding Palette**
- **Input Palette**
- **Navigation Palette**
- **Report Viewer**

  The Report Viewer component allows embedding reports from the Reporting Module into a Perspective views.

Symbols Palette

View Object

This is the view itself. A view has properties that control the view size, defines how tags are dropped into a view, …

Perspective - Chart Palette

Chart Components

Charts allow you to display and show off your data in a graphical way.

The following is a complete list of Chart components that give you various options for displaying data, and a link pointing to a page containing the component's description, properties, and an example of how to configure it.

Chart Range Selector

The Chart Range Selector component is a small recreation of a chart that operators can use to select a time range…

Gauge

The Gauge component in Perspective provides a way to show realtime values in a range as they change.

Pie Chart

A Pie Chart displays a list of named items, each of which has a value that is part of a total.

Power Chart

The Power Chart collects and displays data based on the pens that have been configured on the chart.

Simple Gauge

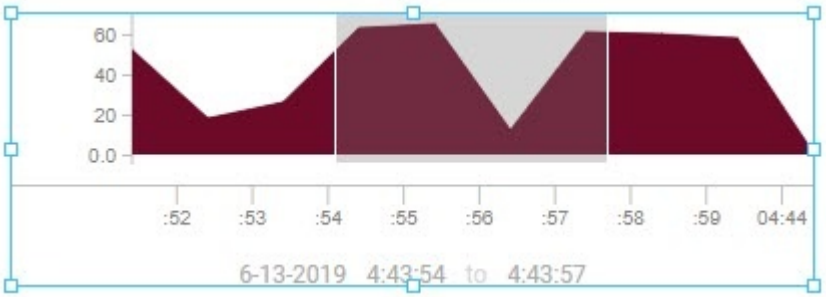
The Simple Gauge component in Perspective provides a way to show realtime values in a range as they change.

Time Series Chart

The Time Series component provides an efficient way to visualize data from a variety of different data sources as …

XY Chart

Perspective - Chart Range Selector



Component Palette Icon:



The Chart Range Selector component is a small recreation of a chart that operators can use to select a time range based on seeing the existing data. This component complements the **Time Series Chart** component, and should always be used with a chart. Its features include:

- Zoom and pan in/out via mouse wheel interaction.
- Click-and-drag brush range selection and panning.
- Start and End property values that are updated as the brush range changes. These properties can govern the start/end points of data queries to return a dataset.
- Time range showing the overall range of the data being displayed by the brush. (The range updates as the brush is updated.)
- Simple display customization for the axes, baselines, markers, and the overall chart data appearance.
- Label properties have their own dedicated styling properties, such as color and size.

User Interaction

Interaction	Description
Zoom	The user can zoom in and out on the Chart Range Selector, but can not zoom out past its standard level of zoom.
Pan	The user can pan across the Chart Range Selector.
Refresh	The Chart Range Selector will not refresh its time range if it is zoomed in.
Pinch Zoom	On a mobile device, the user can pinch-zoom the Chart Range Selector. Zooms must originate from within the boundaries of the displaying chart data.
Brushes	On a mobile device, the user can draw brushes in the Chart Range Selector via a single touch point (multiple touch points will allow zooming to occur). Brushes can be moved in the Chart Range Selector via a single touch point.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
enablePanZoom	Allow the chart to be panned and zoomed. The chart cannot be zoomed out past its base range.	value: boolean
data	Objects that are the data source for the chart (required). Data can be an object containing a time entry and value entries (all must be numbers) (required). Each value entry must be labeled with the column name to which it corresponds. Data can also be an array containing value entries (all must be numbers). Each value entry consists of a timestamp (which must be the first value) and one or more values that were captured at that time. Finally, data can also be in the form of a dataset, for example, the data property can be bound to a Tag History binding to display either realtime data, or historical data (via start and end dates).	object or array
selectedRange	The start and end points of the selected range (required). This property is updated as you interact with the brush. • **start**: A Unix timestamp in milliseconds. Value is numeric. • **end**: A Unix timestamp in milliseconds. Value is numeric.	object
brushRange	An object used to control the display of the date/time range values at the bottom of the component. • **visible**: Whether the brush range is visible. Value is boolean. • **dateFormat**: The date format of the range using a MomentJS date string. Value is a string dropdown. • **timeFormat**: The time format of the range using a MomentJS time string. Value is a string dropdown.	object
timeAxis	This property provides settings for the X Axis. This property uses the same configuration as the timeAxis property of the Time Series Component. • **visible**: The visible state of the the axis. Value is boolean.	object

Name	Description	Property Type
	• **tickCount**: The number of ticks on the axis (as a multiple of 2, 5, or 10). Value is numeric. • **height**: The height of the axis. Value is numeric. • **color**: The color of the axis. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. Value is string. • **tick**: The configuration of the ticks on the axis. Click to see **tick** properties. • **grid**: Configuration for gridlines to display on this axis. ◦ **visible**: Visible state of the gridlines. Gridlines are shown only for axes that connect directly to the chart. Any satellite axes will display their tick configurations instead of gridlines. Value is boolean. ◦ **color**: Color of the gridlines. ◦ **opacity**: Opacity of the gridlines. Value is numeric. ◦ **dashArray**: Dashed appearance of the gridlines. Value is numeric. ◦ **style**: Style for the gridlines. Full menu of **style options** is available. You can also specify a **style class**. • **style**: A style object containing properties which are applied to the horizontal line of the axis. Any property that would apply to an SVG line element can be used here. See also **style options**.	
yAxis	An object used to control the display of the Y Axis. This component was built with the assumption that this axis may be secondary information, so it can be de-emphasized, if desired. • **visible**: The visible state of the the axis. Value is boolean. • **width**: The width of the axis, in pixels (required). Value is numeric. • **label**: The configuration of the Y axis label. ◦ **visible**: Whether or not the label is visible. Value is boolean. ◦ **text**: The text for the label. Value is string.	object

Name	Description	Property Type
	◦ **offset**: Offset the Y axis label from its default position. This allows you to fine tune the label location, which may be necessary depending on the scale and how much room the tick labels take up. This may be positive or negative. Value is numeric. ◦ **font**: The settings for the label's font. ◦ **style**: Custom CSS styles to apply to the Y axis label. Any style that applies to an SVG text element can be used. See also style options. • **tick**: The configuration for the ticks drawn on the axis. Click to see **tick** properties. • **grid**: Configuration for gridlines to display on this axis. ◦ **visible**: Visible state of the gridlines. Gridlines are shown only for axes that connect directly to the chart. Any satellite axes will display their tick configurations instead of gridlines. Value is boolean. ◦ **color**: Color of the gridlines. ◦ **opacity**: Opacity of the gridlines. Value is numeric. ◦ **dashArray**: Dashed appearance of the gridlines. Value is numeric. ◦ **style**: Style for the gridlines. Full menu of style options is available. You can also specify a **style class**.	
areaStyles	An object providing default style to the chart trends as a whole. • **colorScheme**: A Color Brewer color scheme to use on the series. See **ColorBrewer2.org** for available color schemes. Value is string. • **colors**: A list of colors to apply to the columns (in order) for each trend. If these values are provided, they will override the value provided for the colorScheme. Value is string.	object
style	Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You	object

Name	Description	Property Type
	can also specify a **style class**.	

timeAxis.tick

Name	Description	Property Type
color	The color of the ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	value: string
label	The configuration of the label drawn on the tick. • **angled**: If set to true, the tick labels will be angled rather than horizontal. Value is boolean. • **format**: The date/time format displayed by each tick using a MomentjS date string (**https://momentjs.com/**). Options are: Auto, Millisecond [638], Second [:35], Hour Minute [8:15], Hour with Meridiem [8 AM], Day of Week, Month, and Hour [Monday 2nd, 08 AM], Abbreviated Day of Week and Month [Mon 2nd], Abbreviated Month and Day of Month [Jan 2nd], Full Month [January], Abbreviated Month and Year [Jan 20], Full Year [2020], [3-2-2020 8 :15:35], [2020-3-2 8:15:35], Unix Millisecond Timestamp [1563464737269] , Unix Timestamp [1563464737]. Default is “Auto”, where the property attempts to figure out the best format. Value is string. For a listing of suggested formats, refer to **https://momentjs.com/docs/#/parsing/string-format/**. Refer to the Examples section at the bottom of this page to see the **Unix Example: Notation**. • **font**: The settings for the label's font. ◦ **color**: Color of the label text. Value is string. ◦ **size**: The font size, in pixels, of the label text. Value is numeric. • **color**: Color of the label text. Value is string. • Style: Custom CSS styles to apply to the tick labels. Any style that applies to an SVG text element can be used. See also **style options**.	object
style	Custom CSS styles to apply to the ticks. Any style that applies to an SVG line element can be used. See also **style options**.	object

yAxis.tick

Name	Description	Property Type
color	The color of the ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	value: string
count	Number of ticks to display on the Y axis. Default value is Auto. When the tick count is Auto, Y axis ticks will be added/removed based on the height of the chart, and the tick spacing is adjusted around whole numbers.	value: numeric or string
label	The settings for the label on the tick. • **format**: Sets the numeric format for the tick label. The value must be a valid **D3 Format** value. Default value is Auto. When the tick label format is Auto, the D3 format ~f is used. This creates a fixed point that will trim any insignificant trailing zeroes. The Y axis supports numeric locale formatting. Numeric values are automatically formatted based on the session locale and the tick.label.format property. Value is string. • **font**: The font style for the label. ◦ **color**: The color of the label text. Value is string. ◦ **size**: The font size, in pixels, of the label text. Value is numeric. • **style**: Custom CSS styles to apply to the baseline label. Any style that applies to an SVG text element can be used. See also style options.	object
style	Custom CSS styles to apply to the ticks. Any style that applies to an SVG text element can be used. See also style options.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1 - Using a Format Property in a prop.timeAxis.tick.label

The new format property represents the preferred date/time format for the timeAxis property. You can enter any preferred date/time format as defined by **https://momentjs.com/docs/#/parsing/string-format/**. Below is a listing of suggested formats and how they can be used.

Unix Example : Notation

```
"Millisecond [638]": "SSS",
"Second [:35]": ":ss",
"Hour Minute [8:15]": "h:mm",
"Hour with Meridiem [8 AM]": "h A",
"Day of Week, Month, and Hour [Monday 2nd, 08 AM]": "dddd Do, hh A",
"Abbreviated Day of Week and Month [Mon 2nd]": "ddd Do",
"Abbreviated Month and Day of Month [Jan 2nd]": "MMM Do",
"Full Month [January]": "MMMM",
"Abbreviated Month and Year [Jan 20]": "MMM YY",
"Full Year [2020]": "YYYY",
"[3-2-2020 8:15:35]": "M-D-YYYY h:mm:ss",
"[2020-3-2 8:15:35]": "YYYY-M-D h:mm:ss",
"Unix Millisecond Timestamp [1563464737269]": "x",
"Unix Timestamp [1563464737]": "X"
```

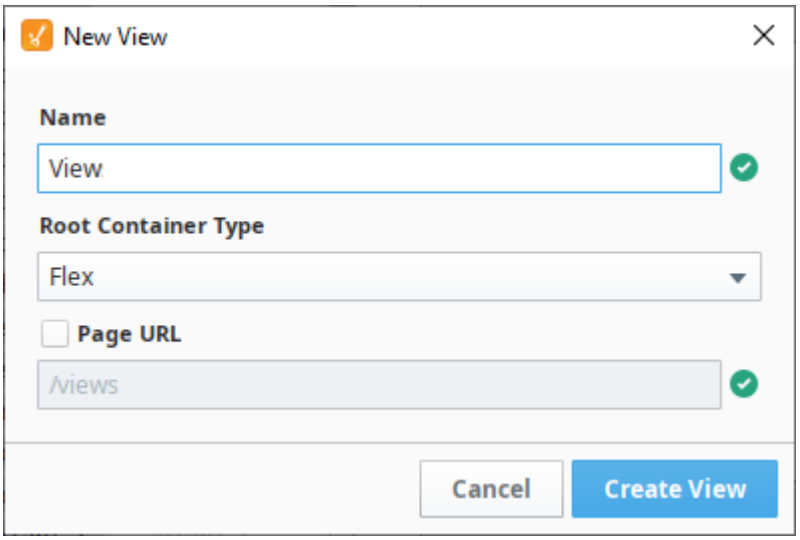
Example 2 - Using the Chart Range Selector

To begin using the Chart Range Selector, a Time Series Chart with trend data will be needed. This example shows how to configure the Chart Range Selector.

1. Begin by **configuring tag history** on a Tag of your choice.

2. From the Perspective section of the **Project Browser** on your Designer, right click on the **Views** folder and select **New View**... to create a new view.

3. This will bring up the New View window. Give your view a name and select the **Flex** Root Container Type. The Page URL setting will remain unchecked for this example.



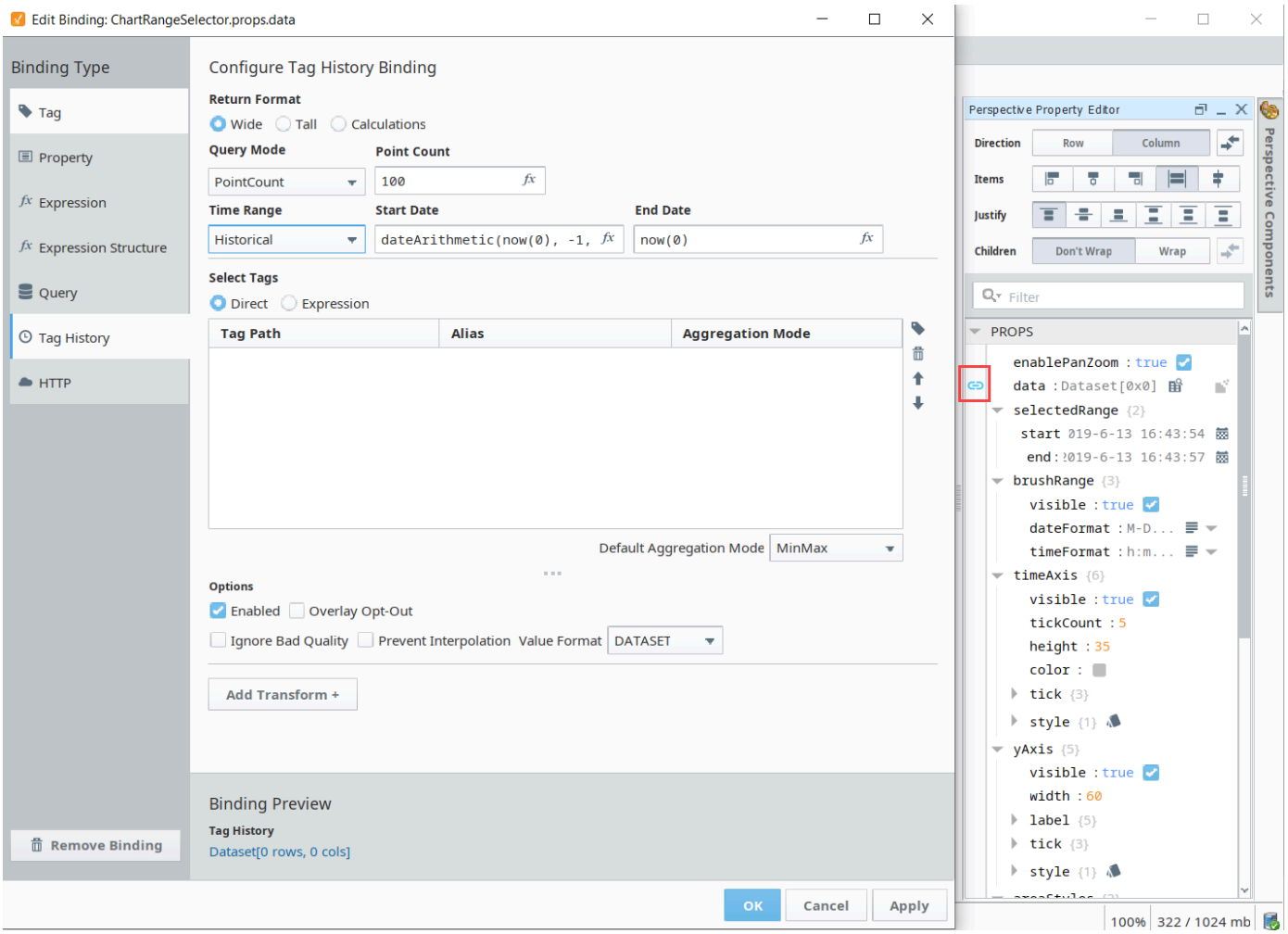
4. From the PerspectiveComponent Palette, drag and drop a **Time Series Chart** onto your newly created view. In the **Position** category of the Property Editor, set the **position.grow** property to **'1'**.

5. Drag and drop a Chart Range Selector component onto your view. Go to the **Postion** category and set the Chart Range Selector's **position.grow** property to **'1'** .

6. With the Chart Range Selector selected, go to the **data** property, click on the binding icon to bring up the Binding Editor window and select the **Tag History** binding type as shown in the image below.

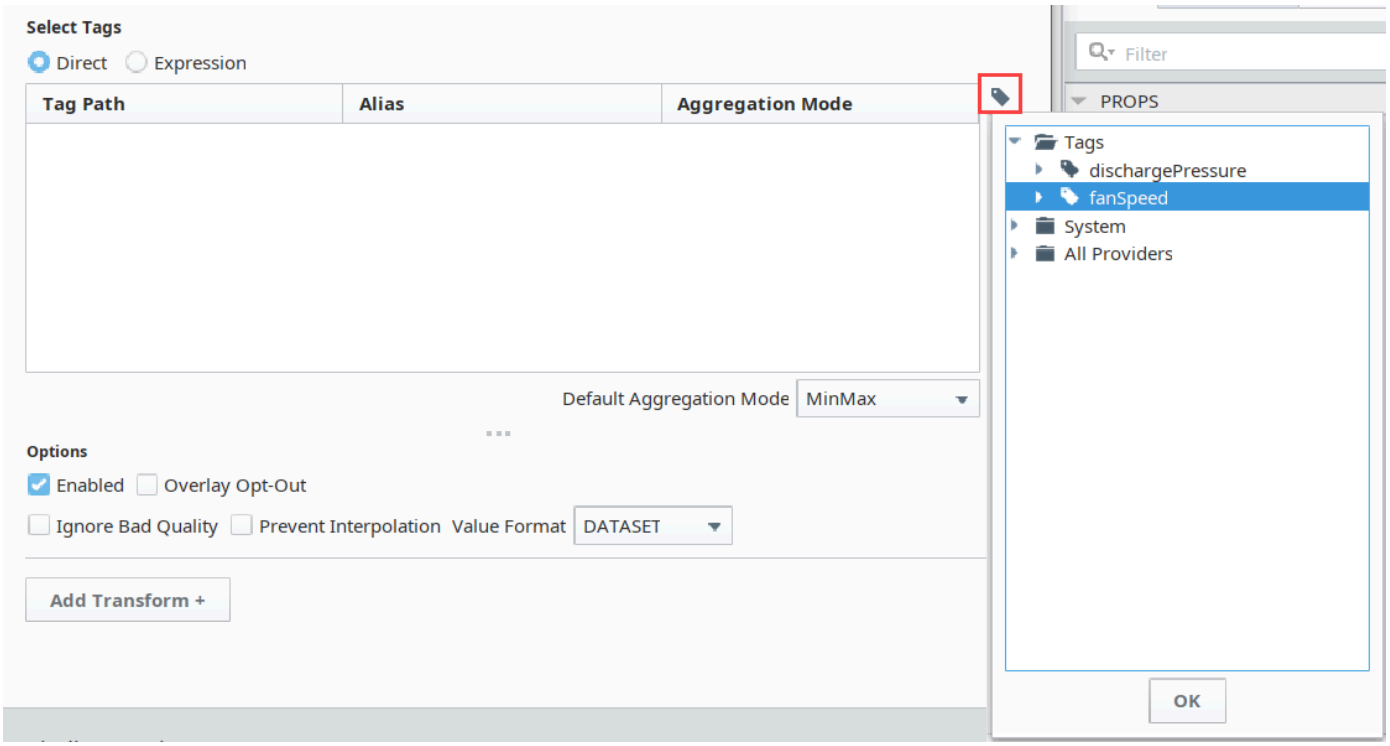
7. Set the **Time Range** to **Historical**. We'll configure the binding to span the last one hour of historical data by making the following changes:

    - Start Date: `dateArithmetic(now(0), -1, 'hour')`
    - End Date: `now(0)`

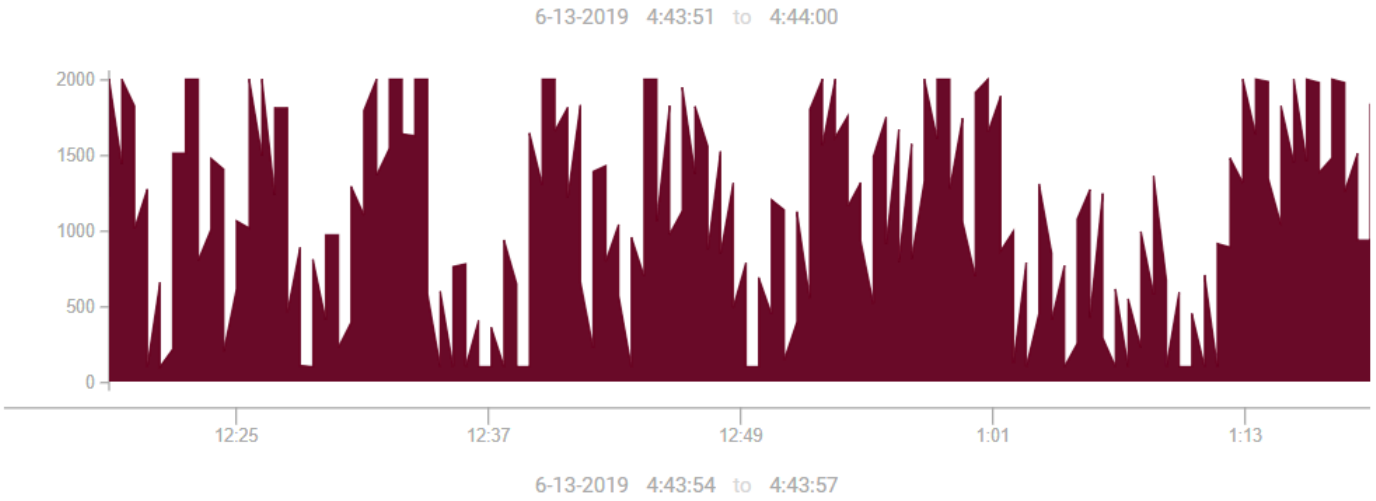


8. Under the Select Tags section, click on the **Tag** icon and use the Tag browser to drill down to the Tag you created in Step 1

9. Click **OK**.

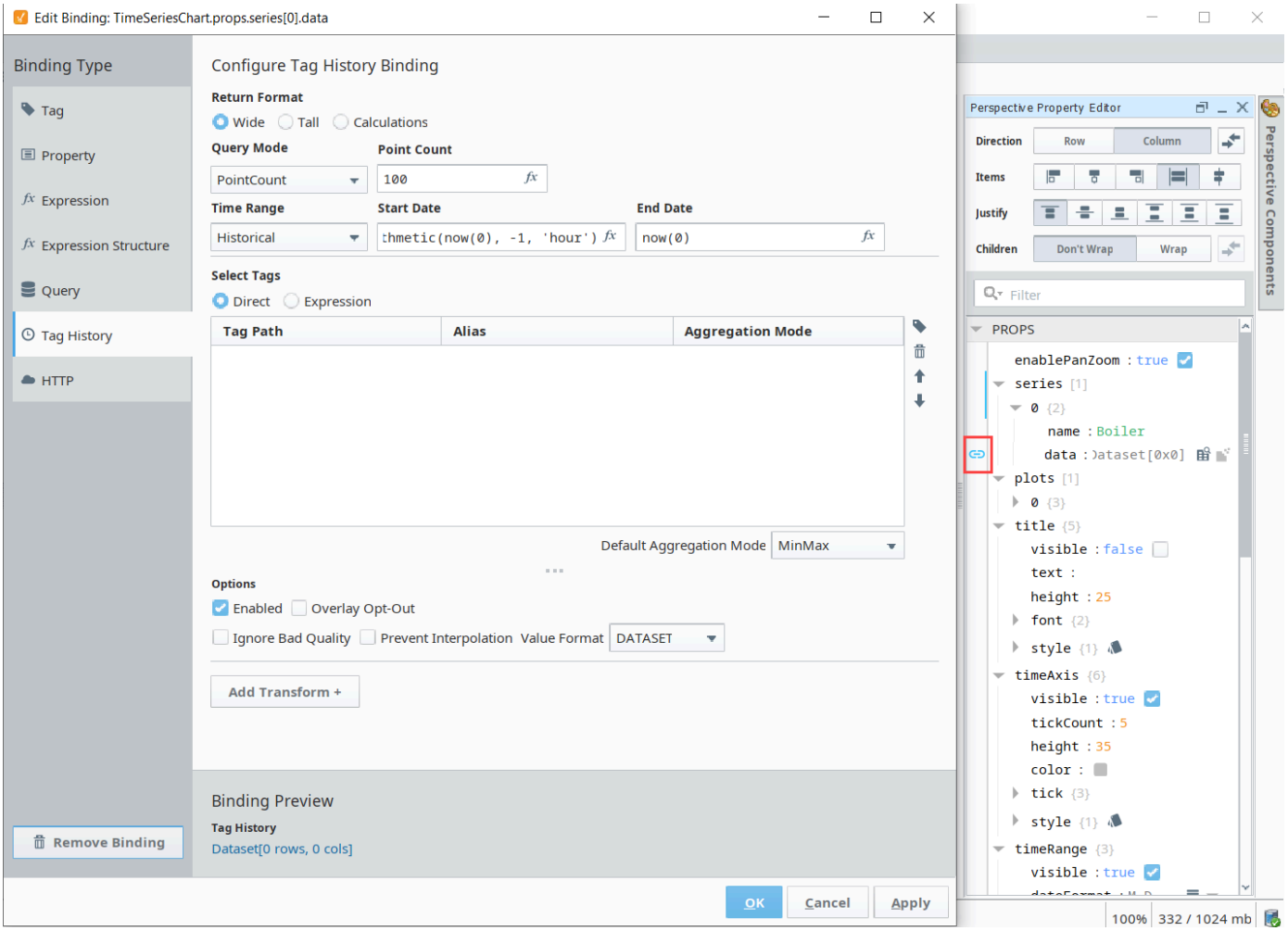


10. Click **OK** on the Binding Editor window to accept the binding changes. You should now have a Chart Range Selector displaying the last 1 hour of historical data for your Tag created in Step 1.



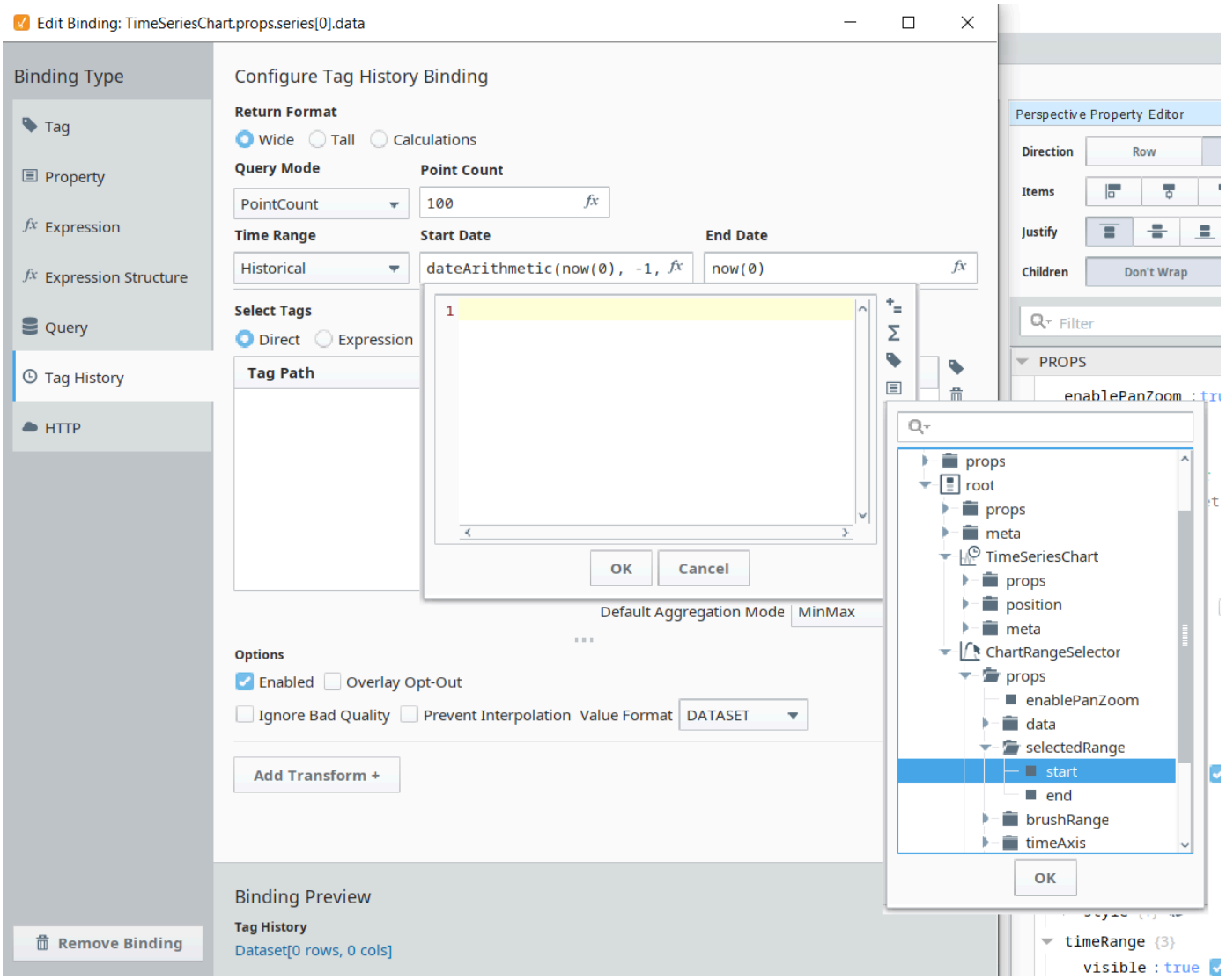
11. Now select your Time Series Chart and from the Perspective Property Editor, and click on the **binding** icon for the **series[0].data** property to open the Binding Editor window.

12. Select the **Tag History** binding type as shown in the following image.



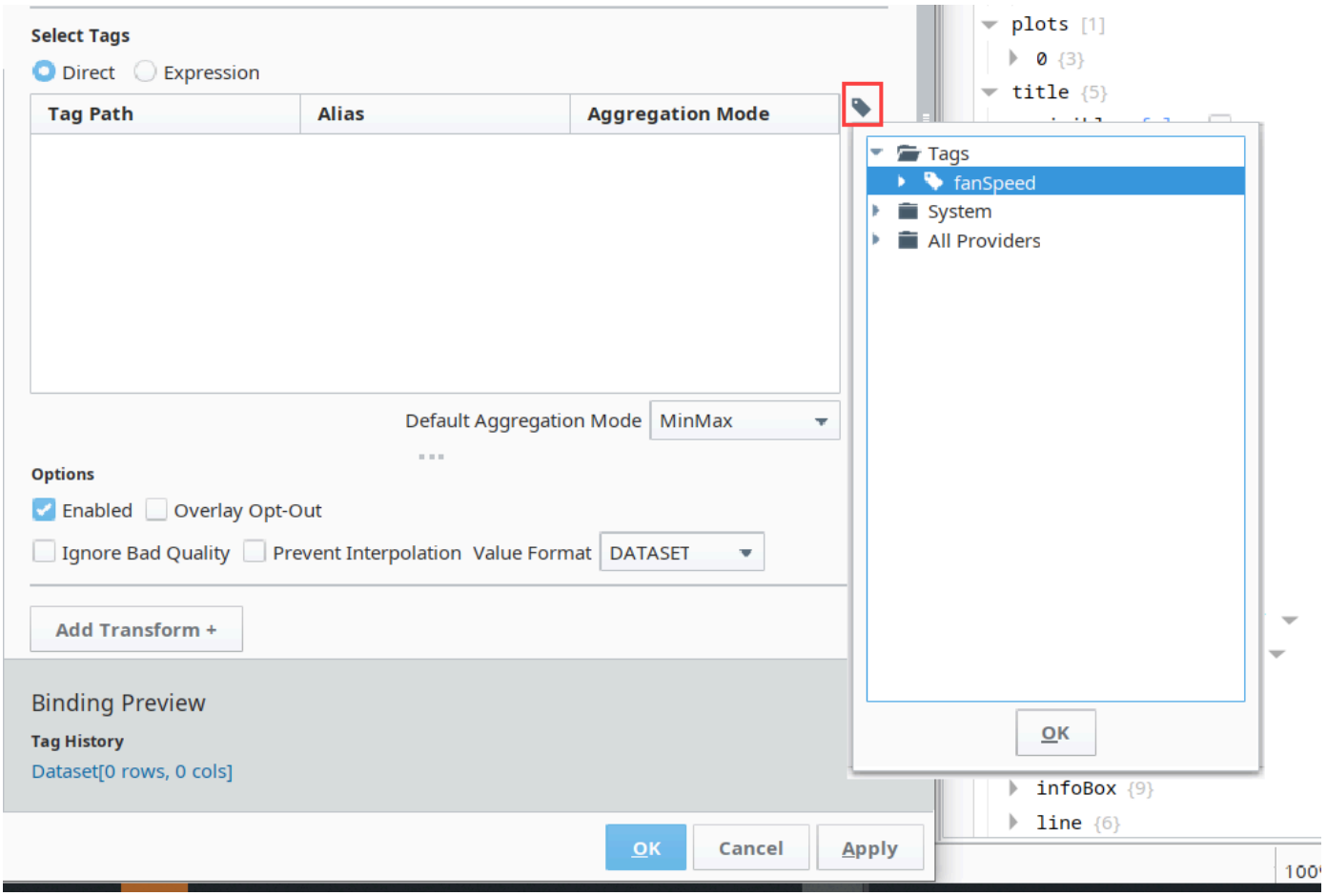
13. Set the Time Range to **Historical**.

14. The Start Date needs to have a property binding configured pointing to the Chart Range Selector's **props.selectedRange.start** property as in the image below.

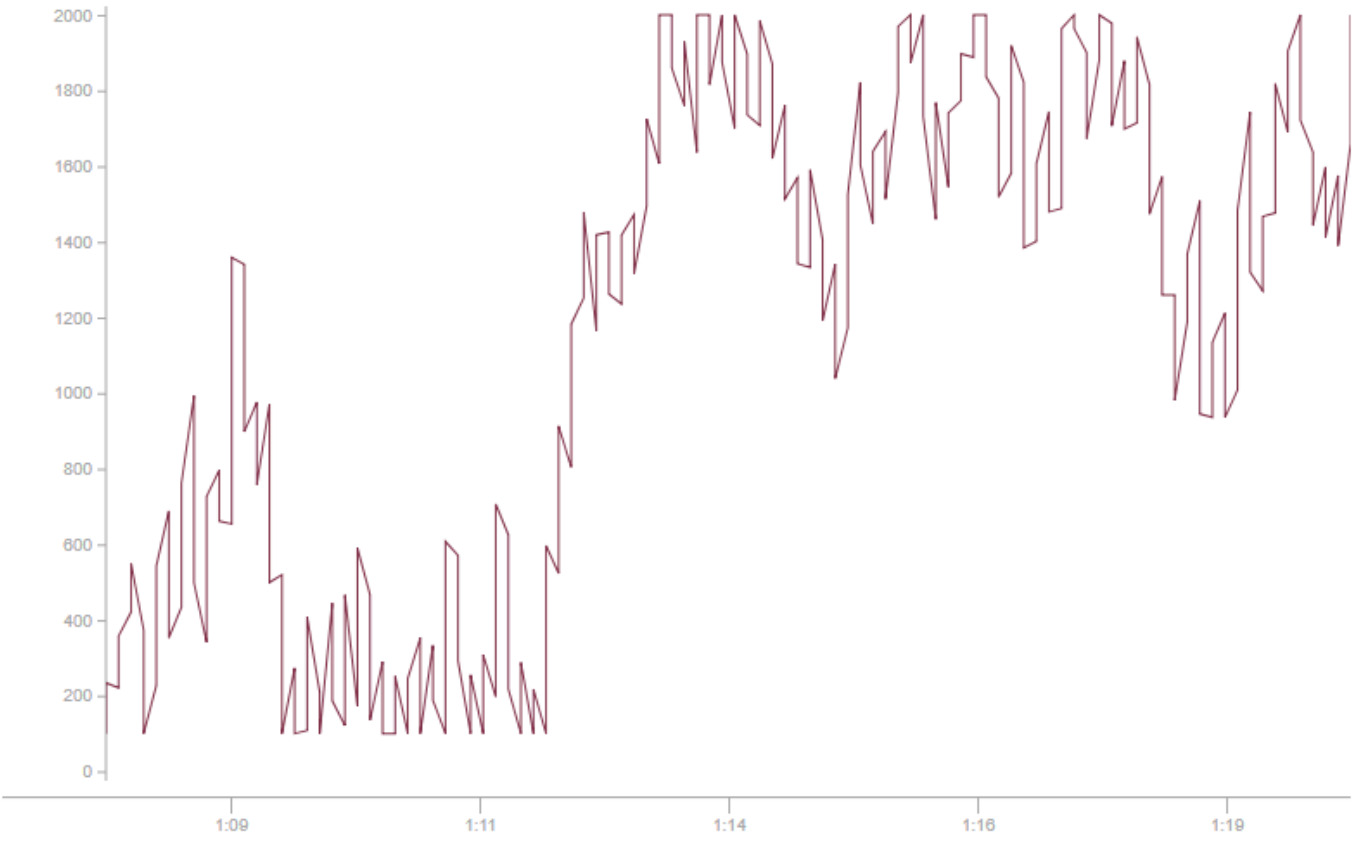


15. Similarly, the **End Date** needs to have a property binding configured pointing to the Chart Range Selector's **props.selectedRange.end**.

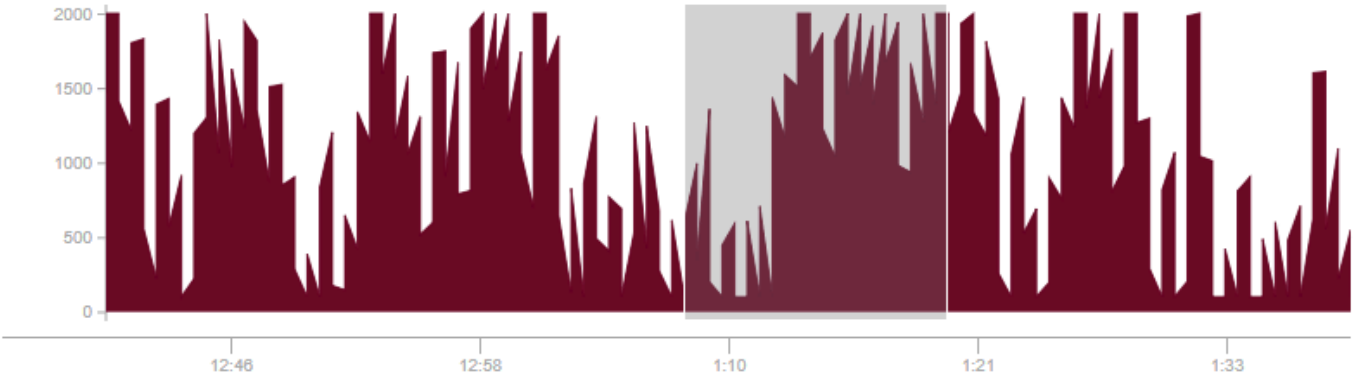
16. For the Tag History Binding configuration, click on the **Tag** icon on the right of the Select Tags table and drill down to the Tag from Step 1, then click **OK**.



17. After clicking OK and accepting the binding configurations on the Time Series Chart, you will be able to use your Chart Range Selector to select what data you want on your Time Series Chart to display. Simply drag and re-size the Chart Range Selector's brush section as shown below.

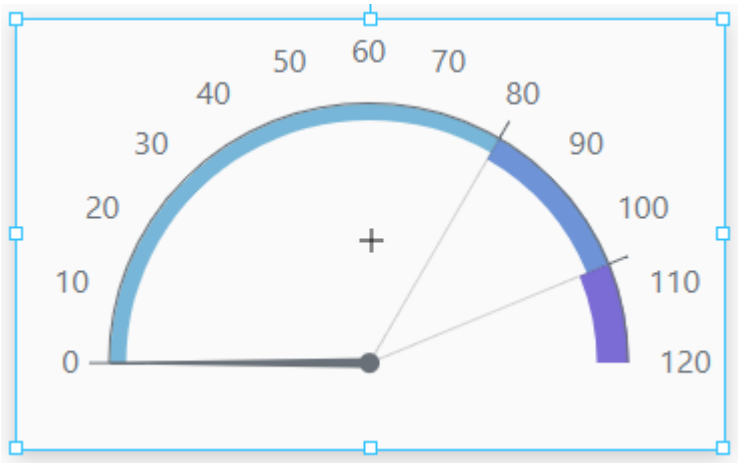


6-10-2020 1:07:55 to 1:20:20



6-10-2020 1:07:55 to 1:20:27

Perspective - Gauge



Component Palette Icon: Gauge

The Gauge component in Perspective provides a way to show realtime values in a range as they change. The gauge can have one or two axis. It is fully customizable in its appearance, from colors, line widths, needle length, radius, and more.

The Gauge component has three pre-configured **variants**:

- Half Circle - Default layout with a half-circle gauge.
- 3/4 Circle - Layout with a 3/4 circle gauge.
- Full Axis - Layout with a full axis gauge. For an example of each variant, see the examples section below.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	Numeric value for the gauge to display. Default is 0.	value: numeric
secondaryValue	Optional secondary value for the gauge to display on a second axis. Default is 0.	value: numeric
startAngle	Radial position for the start of the gauge's axis. Default is 180.	value: numeric
endAngle	Radial position for the end of the gauge's axis. Default is 360.	value: numeric
outerAxis	Sets the values for an outer axis on the gauge. Click to see the **outerAxis** properties.	object
innerAxis	Sets the values for an inner axis on the gauge. Click to see the **innerAxis** properties.	object
backgroundColor	Color applied as a background within the gauge. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	color
animate	Whether needle should be animated in a sweeping motion when value changes. Default is false.	value: boolean
reverseScale	If true, the gauge will reverse the direction from minValue to maxValue on its dial.	value: boolean
style	Sets a style for this gauge. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

outerAxis Properties

Name	Description	Property Type
data	What value this axis and its needle should display. Can be set to the value or secondary value of the gauge.	value: string dropdown
show	Whether the outer axis is displayed. Default is true (show).	value: boolean
minValue	Minimum gauge value for this axis. Default is zero (0).	value: numeric
maxValue	Maximum gauge value for this axis. Default is 120.	value: numeric
width	Width of the line (in pixels) that represents the outer axis.	value: numeric
color	Color of the arc line that represents the outer axis. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	color
percentRadius	Radius of the outer axis, as a percentage of the total chart radius.	value: numeric
ranges	Zones defined on the gauge arc line with a unique color. Array values always start at 0 and increment. • **start**: Value at which this range starts. Value is numeric. • **end**: Value at which this range ends. Value is numeric. • **width**: Width of this axis, in pixels. Value is numeric. • **color**: Color to apply to this range of the dial. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value.	array
needle	Settings for the needle on the gauge.	object

Name	Description	Property Type
	• **origin**: Distance from the gauge's center or at which the needle originates, as a percentage of the radius. For example, a value of 0 indicates the needle starts at the center point of the gauge. A value of 50 indicates it starts 50% from the center point. Value is numeric. • **reach**: How far the needle reaches from the center of the gauge towards the outer dial, as a percentage of the radius. For example, a value of 100 indicates the needle will reach all the way to the outer axis. Value is numeric. • **color**: Color of the gauge's needle. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	
tickMarks	Settings for the display of the tick marks on the outer access. • **color**: Color of the tick marks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. • **thickness**: Thickness of the tick marks, in pixels. Default is 1. Value is numeric. • **length**: Length of the tick marks, in pixels. Default is 10. Value is numeric.	object

innerAxis Properties

Name	Description	Property Type
data	What value this axis and its needle should display. Can be set to the value or secondary value of the gauge.	value: numeric
show	Whether the inner axis is displayed on the gauge. Default is false (don't show).	value: boolean
minValue	Minimum gauge value for this axis. Default is zero (0).	value: numeric
maxValue	Maximum gauge value for this axis. Default is 80.	value: numeric
width	Width of the line (in pixels) that represents the inner axis.	value: numeric
color	Color of the arc line that represents the inner axis. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	color
percentRadius	Radius of the inner axis, as a percentage of the total chart radius.	value: numeric
ranges	Number of zones defined on the gauge arc line with a unique color. • **start**: Value at which this range starts. Value is numeric. • **end**: Value at which this range ends. Value is numeric. • **width**: Width of this axis, in pixels. Value is numeric. • **color**: Color to apply to this range of the dial. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	array
needle	Settings for the display of the needle on the inner access. • **color**: Color of the gauge's needle. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	object

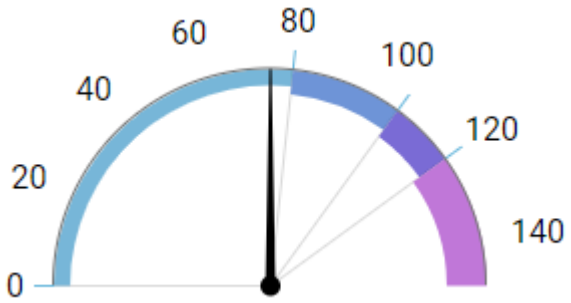
Name	Description	Property Type
	• **origin**: Distance from the gauge's center at which the needle originates, as a percentage of the radius. For example, a value of 0 indicates the needle starts at the center point of the gauge. For example, a value of 50 indicates it starts 50% from the center point. Value is numeric. • **reach**: Length of the needle in percentage. For example, a value of 100 indicates the needle will reach all the way to the inner axis. Value is numeric.	
tickMarks	Settings for the display of the tick marks on the inner access. • **color**: Color of the tick marks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. • **thickness**: Thickness of the tick marks, in pixels. Default is 1. Value is numeric. • **length**: Length of the tick marks, in pixels. Default is 10. Value is numeric.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1 - Fourth Axis



In this example we added a fourth axis (outerAxis.ranges.3) and configured it.

Property	Value
Value	75
props.outerAxis.maxValue	150
props.outerAxis.ranges.3.start	120
props.outerAxis.ranges.3.end	150
props.outerAxis.ranges.3.width	20
props.outerAxis.ranges.3.color	#C077D8

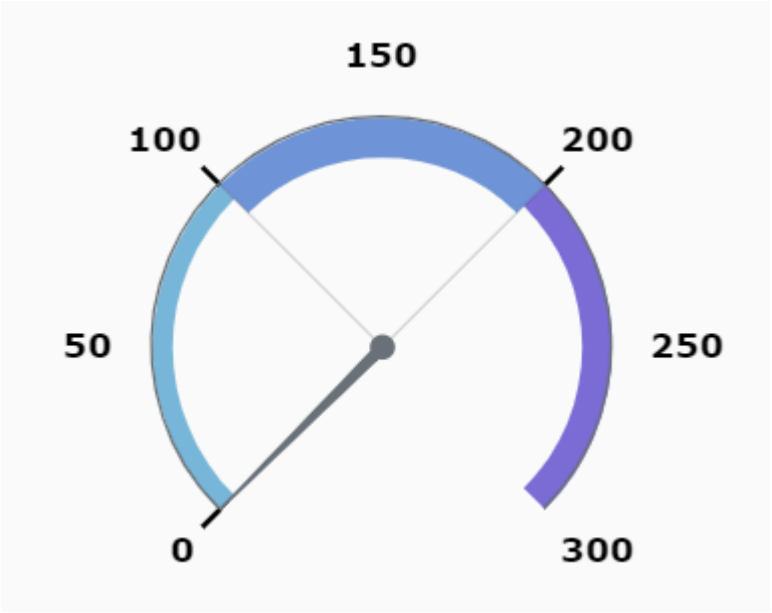
Example 2 - 3/4 Circle Variant



Start with the Gauge 3/4 Circle variant. Set the properties as follows:

Property	Value
props.startAngle	90
props.endAngle	360
props.outerAxis.ranges.0.color	#FFFF00
props.outerAxis.ranges.1.color	#FF8C00
props.outerAxis.ranges.2.color	#D90000

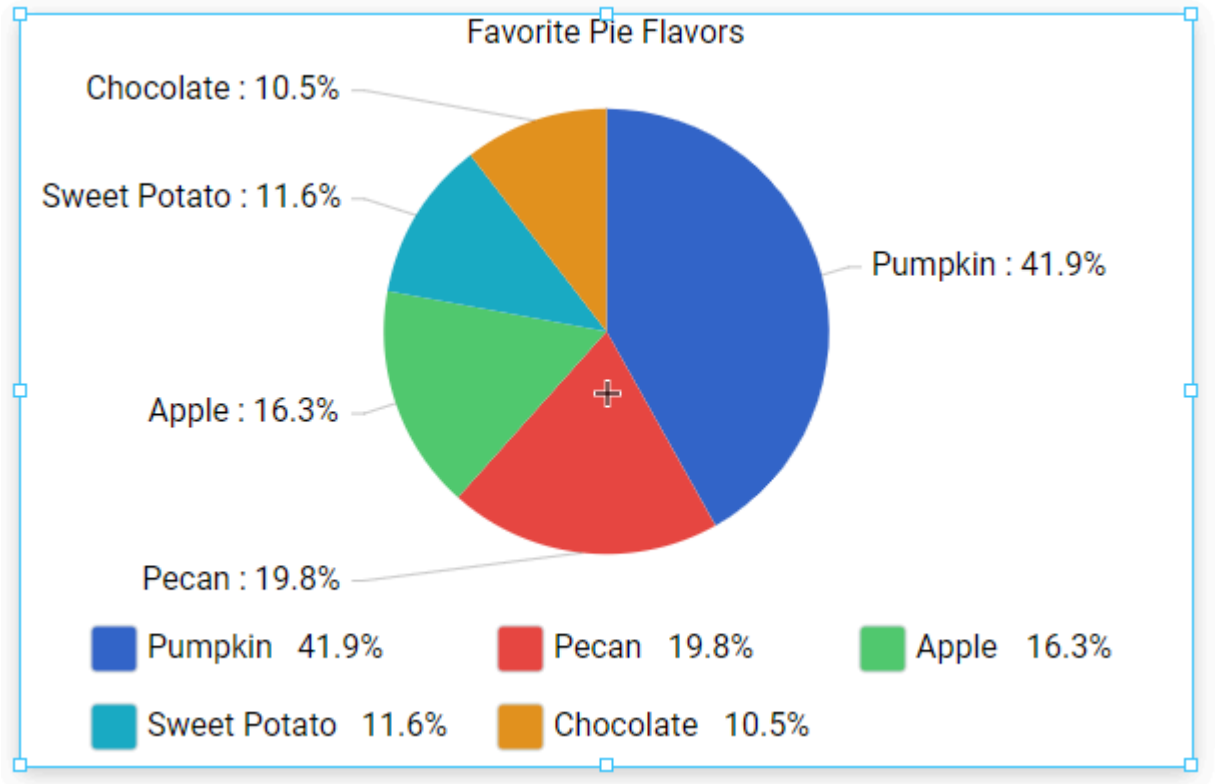
Example 3 - Full Axis Variant



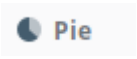
Start with the Gauge Full Axis variant. Set the properties as follows:

Property	Value
props.startAngle	135
props.endAngle	405
props.outerAxis.maxValue	300
props.outerAxis.ranges.0.end	100
props.outerAxis.ranges.1.start	100
props.outerAxis.ranges.1.end	200
props.outerAxis.ranges.2.start	200
props.outerAxis.ranges.2.end	300
props.outerAxis.tickMarks.color	#000000
props.outerAxis.tickMarks.thickness	2
props.style.color	#000000
props.style.fontFamily	Verdana
props.style.fontSize	12
props.style.fontWeight	bold

Perspective - Pie Chart



Component Palette Icon:



A Pie Chart displays a list of named items, each of which has a value that is part of a total. The total is the sum of the value of each item. The key to the Pie Chart component is the data property, which contains the items that will be displayed as pie wedges. It is fully customizable in its appearance, from colors, line widths, text styles, and more.

The Pie Chart component has two pre-configured **variants**:

- Flat Chart - Default component described above.
- Three-Dimensional Chart - Component pre-set with three-dimensional pie wedges.

Properties

Name	Description	Property Type
data	Data source for the chart. Each object within an array defines the name and value for a single pie section.	array
colors	Colors that correspond to each pie section, respective of order in data. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	array
title	Name to display for this chart.	value: string
tileColor	Color of the title. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	color
valueFormat	Label and legend value format configuration. • **showPercentSymbol**: Whether to show the percent symbol next to the percent value. Value is boolean. • **showValueAsPercent**: Whether to show the value as percent. Value is boolean.	array
showLabels	Whether to show labels for each section of this chart. Default is true (show).	value: boolean
labels	Settings for the labels. • **showName**: Whether to show the name on the label. Value is boolean. • **showValue**: Whether to show the value on the label. Hiding values will disable any value formats set. Value is boolean. • **bent**: Bend labels around chart slices. Default is false. Value is boolean. • **align**: Whether the labels should be aligned in vertical columns. Value is boolean.	object

Name	Description	Property Type
	• **inside**: Settings for showing labels inside of the chart slices instead of outside. ◦ **enabled**: Value that determines whether to show the labels inside of the chart slices instead of outside, based on if the value percentage is below the percentLimit threshold. Value is boolean. ◦ **radius**: Distance in percentage towards center of Pie Chart while inside is enabled. 0 represents outside edge while 100 would be directly in the middle. Value is numeric. ◦ **color**: Label color for labels while they are displayed inside the chart. ◦ **percentLimit**: Value that determines at what value percentage to place label on outside of chart instead of inside. Value is numeric. • **wrap**: Label text wrapping configuration. Ability to wrap long labels. ◦ **enabled**: Enables label text wrapping. Default is false. Value is boolean. ◦ **maxWidth**: The maximum allowable label width, in pixels. • **color**: Color of the labels. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	
tooltipFormat	This value determines the format of the tooltips. Click to see tooltipFormat **Placeholder Objects** and **Formatting Options**.	Value is string.
showLegend	Whether to show a legend for this chart. Default is true (show). Value is boolean.	
legend	Settings for legend. • **fontSize**: Font size for legend labels. Value is numeric. • **icon**: Settings for the icon on entries in the legend. ◦ **height**: Height value of legend icon. Value is numeric.	object

Name	Description	Property Type
	◦ **width**: Width value of legend icon. Value is numeric. ◦ **enabled**: Value that determines whether to show the legend icons or hide them. Value is boolean. • **position**: Aligns legend to specified direction. Value is string.	
legendLabelColor	Color of the legend labels.	color
cutoutRadius	Percent of total radius to cut out of center of chart. If greater than zero, the chart becomes ring-style instead of pie.	value: numeric
selection	An object that contains selection related properties. • **enabled**: Enables selection of pie chart slices. Value is boolean. • **data**: A read-only list of selected pie chart slices.	object
selectOutline	Outline for each section of the pie. • **width**: Width of the border (in pixels) around the pie section. Value is numeric. • **color**: Color of border around each pie section. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. • **opacity**: Opacity of border around each pie section. 0 is fully transparent, 1 is fully opaque. Value is numeric.	object
enableTransitions	Whether the chart has visual transition effects for changes in chart data.	value: boolean
threeDimensional	Whether the chart has depth effect to look three-dimensional.	value: boolean
style	Use styles to customize the visual style of the component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

tooltipFormat Placeholder Objects

The value of this property expects a string, and recognizes certain placeholders objects. The objects are fully detailed in **amChart's documentation**, and we listed some commonly used objects below. Note that each object should be wrapped in a set of braces (`{` and `}` characters).

Object	Description	Example
Category	References the value of the wedge's category (the string value for the wedge), allowing the tooltip to display a string value for the wedge. The value of category is based off the string value for the wedge. Thus, in the Pie Chart Example on this page, category would return the value of the "flavor" property.	`{category}`
Value	Returns the value of the wedge. In addition to just the value of the wedge, additional modifiers can be added to obtain pre-calculated derivative values, such as the sum or largest value. A full list of modifiers can be found in the **amCharts documentation**.	`{value}` `{value.sum}` `{value.high}`

tooltipFormat Formatting Options

In addition to placeholder objects, the tooltip can also be formatted in the following ways. All formatting is placed between a set of brackets (`[` and `]` characters) as a single **block**. One or more formatting options can be place into a single block. When specifying multiple options, each should be delimited by a single space. Example: [bold red]

Option	Description	Example Format	Example Output
Bold	Applies bold formatting to the characters in the block.	`Hello [bold]World`	Hello **World**
Colors	Allows you to specify the color of the text. Colors can be specified as RGB color codes, or named colors.	`Hello [red]World` OR `Hello [#ff0000]World`	Hello World
In-line CSS	Allows you to directly apply CSS attributes.	`Hello [font-style: italic]World`	Hello *World*
Closing Bracket [/]	Terminates the formatting string. Note that this is optional. Style formatting will automatically be terminated by either the next opening style block (`[` character) or the end of the string, whichever comes first.	`[bold]Hello [/]World!`	**Hello** World!
Formatting Functions	In addition to the above, several functions are available to further format the placeholder objects. A full list of functions can be found in the **amChart documentation**.	`{value.formatNumber('###.00')}`	100.00

Component Events

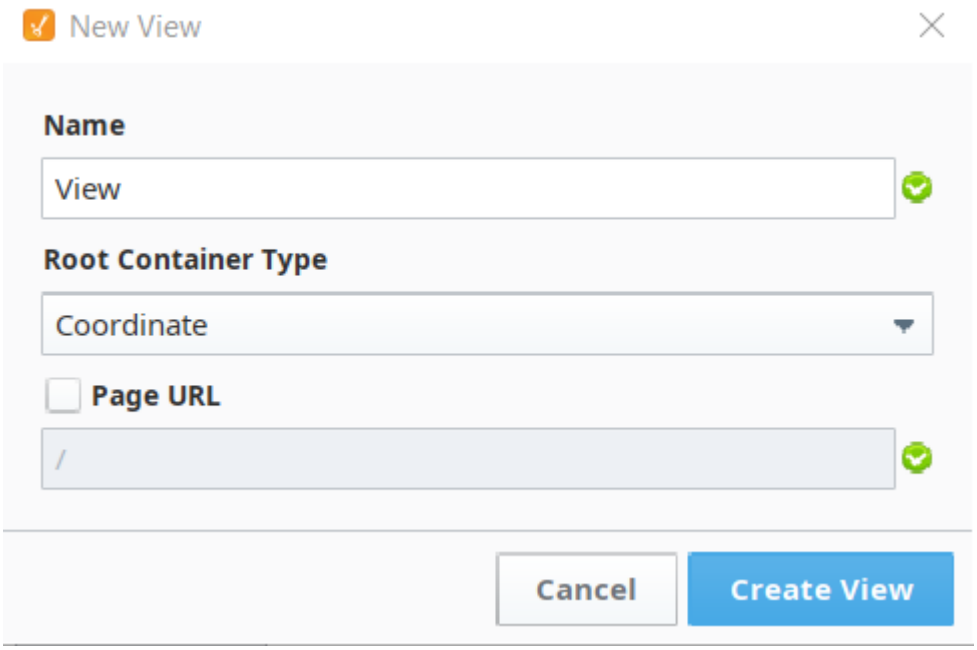
The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

The Pie Chart component can be used for things like inventory tracking. Below is an example that uses a Pie Chart to display the inventory of a local ice cream shop.

1. From the Perspective section of the Project Browser in your Designer, right-click on the Views folder and select **New View...** to create a new view.

2. This will bring up the New View window. Give your view a name and select the Coordinate Root Container type. The Page URL setting will remain unchecked for this example.



3. Drag and drop a Pie Chart from the Perspective Component Palette onto your newly created view.

4. Set the Pie Chart's title property to **Ice Cream Inventory**.

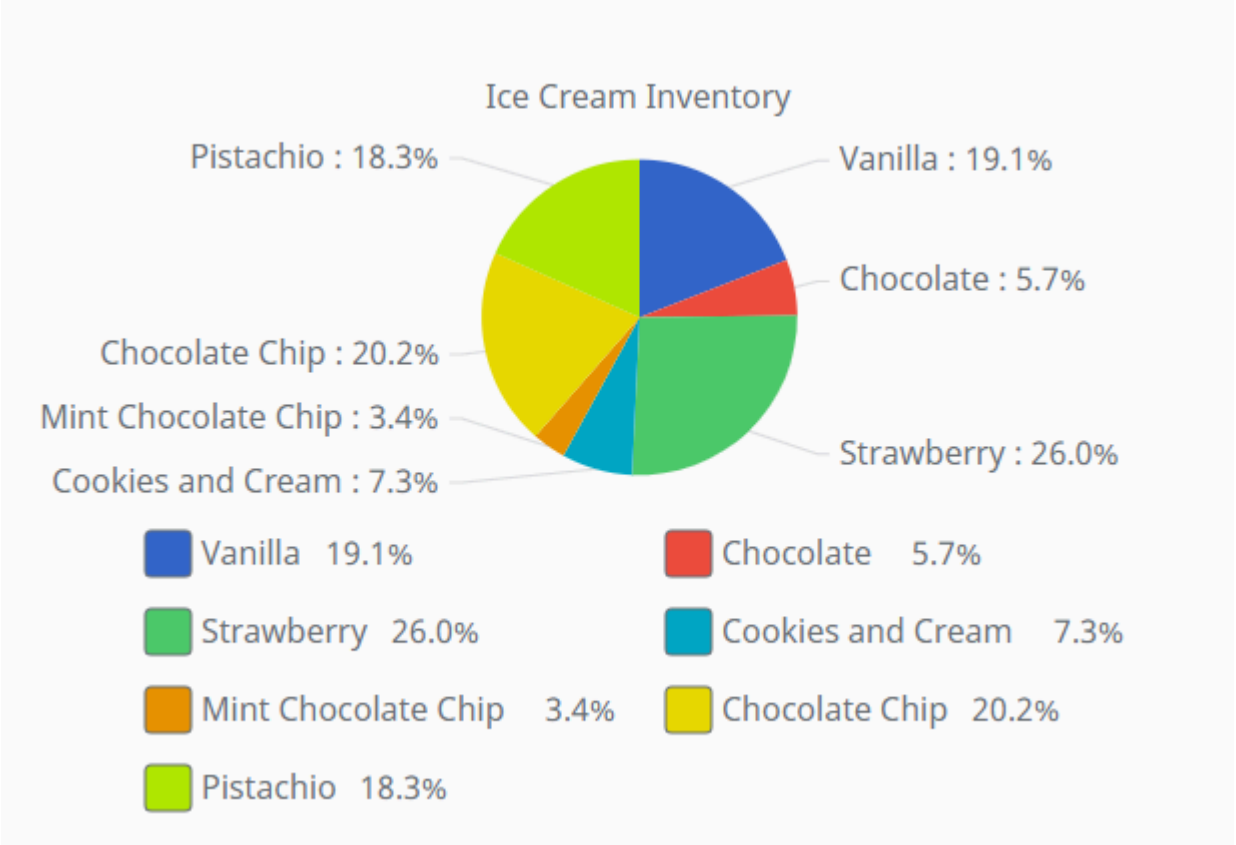
5. Copy the array below and paste it on the Pie Chart's data property.

```
[
  {
    "flavor": "Vanilla",
    "count": "50"
  },
  {
    "flavor": "Chocolate",
    "count": "15"
  },
  {
    "flavor": "Strawberry",
    "count": "68"
  },
  {
    "flavor": "Cookies and Cream",
    "count": "19"
  },
  {
    "flavor": "Mint Chocolate Chip",
    "count": "9"
  },
  {
    "flavor": "Chocolate Chip",
    "count": "53"
  },
  {
    "flavor": "Pistachio",
    "count": "48"
  }
]
```

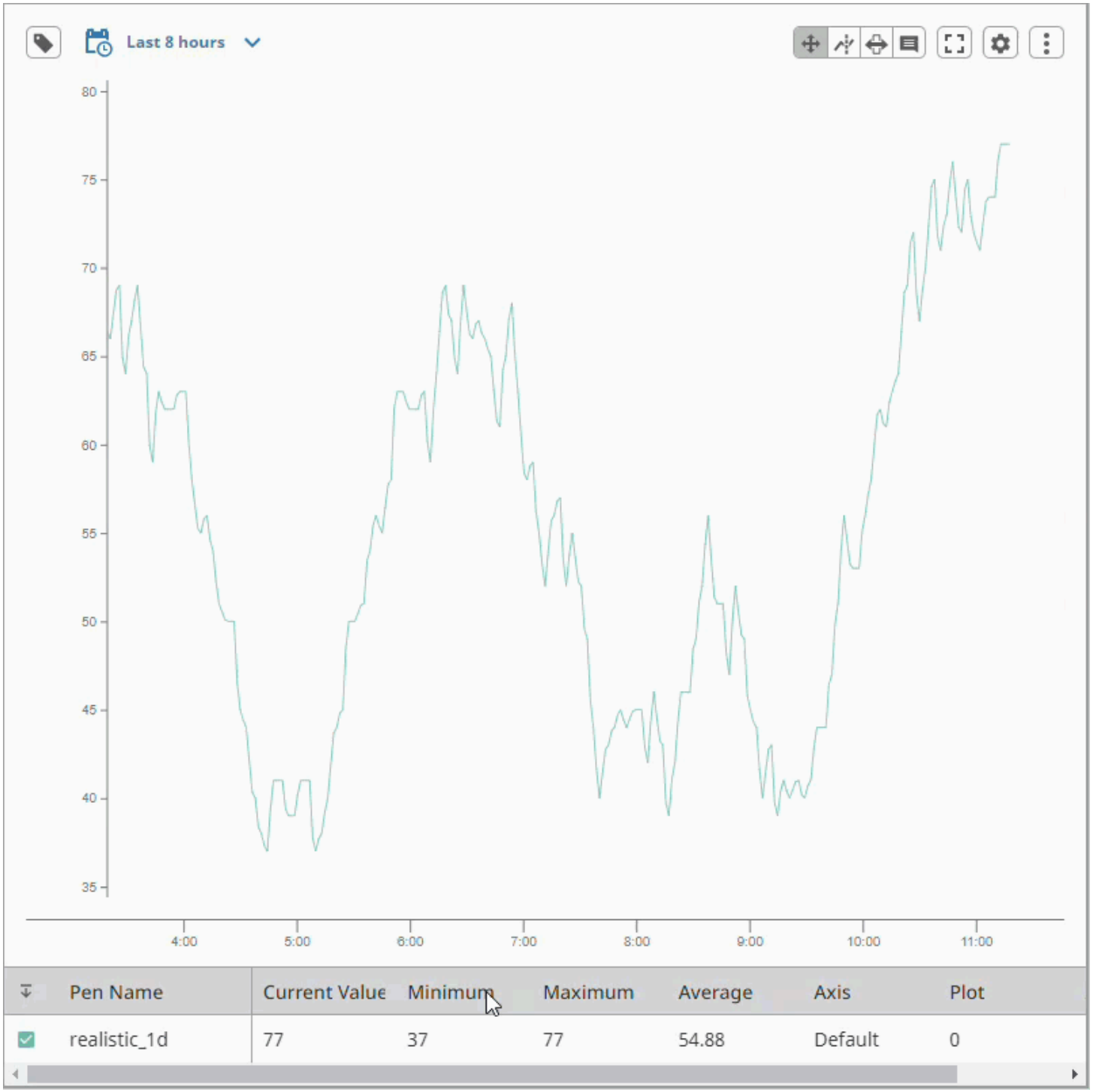
NOTE

The array above is a an array of objects. Each object is a dictionary containing key/value pairs where the keys of each dictionary represent ice cream flavors and ice cream inventory counts with their respective values for each. Value types can vary as you can pass both "50" and 50 as counts and the Pie Chart will still be able to render the data correctly. The Pie Chart can have various forms of data sources. The data source array can be built via scripting following the above format or it can be built using a **query binding** on the Pie Chart's data property. The query used must return two columns in any order where each column represents a string and a numeric value to be rendered by the chart.

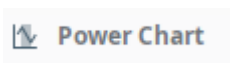
6. Once you have done this, your Pie Chart should accurately represent of the inventory data for the ice cream shop.


Ice Cream Inventory
Pistachio : 18.3%
Vanilla : 19.1%
Chocolate : 5.7%
Chocolate Chip : 20.2%
Mint Chocolate Chip : 3.4%
Cookies and Cream : 7.3%
Strawberry : 26.0%
Vanilla 19.1%
Chocolate 5.7%
Strawberry 26.0%
Cookies and Cream 7.3%
Mint Chocolate Chip 3.4%
Chocolate Chip 20.2%
Pistachio 18.3%

Perspective - Power Chart



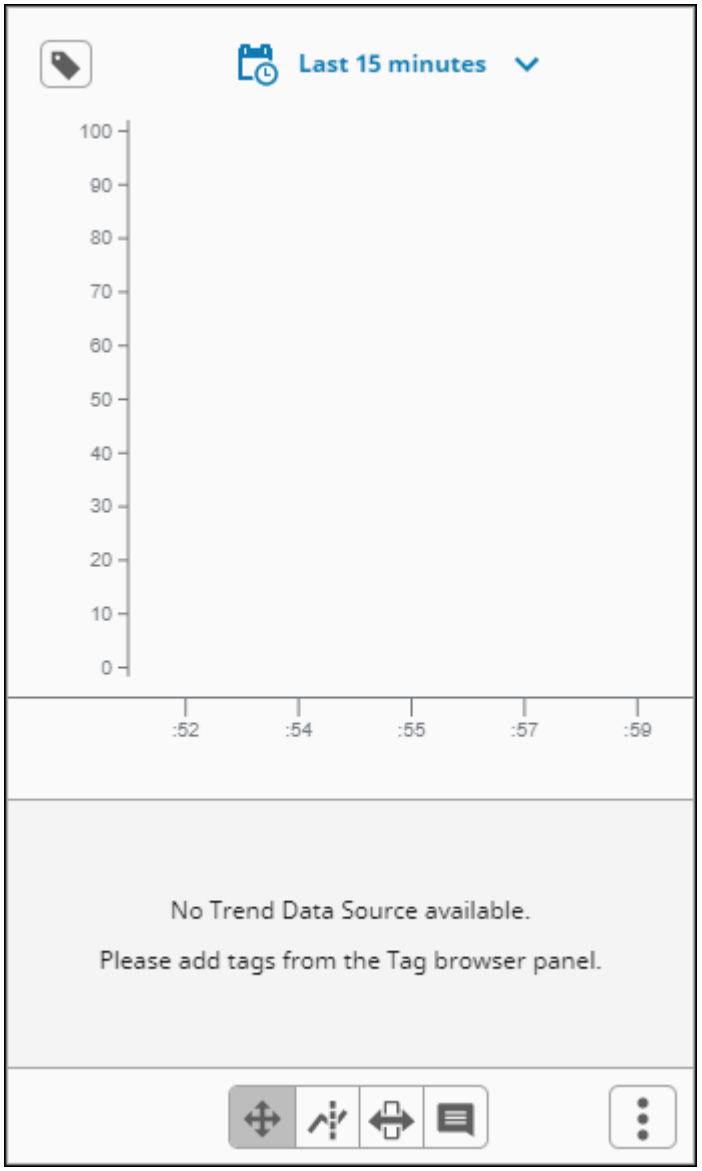
Component Palette Icon:



The Power Chart collects and displays data based on the pens that have been configured on the chart. Users can add or remove pens from the chart to change the underlying data. It is fully customizable in its appearance, from labels, colors, line widths, legend, scroll bars, and text styles.

> **NOTE**
>
> The Power Chart utilizes functionality provided by the **Historian Core** module, and requires a Historian license to function.

The Power Chart has a responsive design and a mobile-optimized display that is different than the standard display. It has a mobile breakpoint of 750px to fit on mobile devices. The mobile breakpoint is configurable on the `responsiveDesignWidth` property. The image below shows how the Power Chart renders on a smaller device.



Component Anatomy

Aside from the chart trending area, the component features Browse Tags, Pen Control, and Chart Settings panels that provide additional functionality.

Browse Tags

The Browse Tags panel allows you to browse for any available historical data, and add it to the chart. It is similar to the Designer's Tag Browser, but this tree reports any records that are accessible from the Historian system, including tables provided by the **DB Table Historian Provider**. There are two ways to add tags to the chart's display:

- Select any nodes (entries with the Tag icon), and click **Add Selected Tags**, which will add a pen to the chart that represents the node that was selected.
- Drag selected nodes onto the chart. You can select multiple items by using Ctrl-click and you'll see a prompt indicating how many tags are selected.

If there's more than one plot in the chart configuration, you will be prompted to choose the plot to add the pen to.

Pen Control

This table shows each pen that's currently on the component, offers aggregates based on the chart's current range, and provides quick actions like hiding the pen and changing its color.

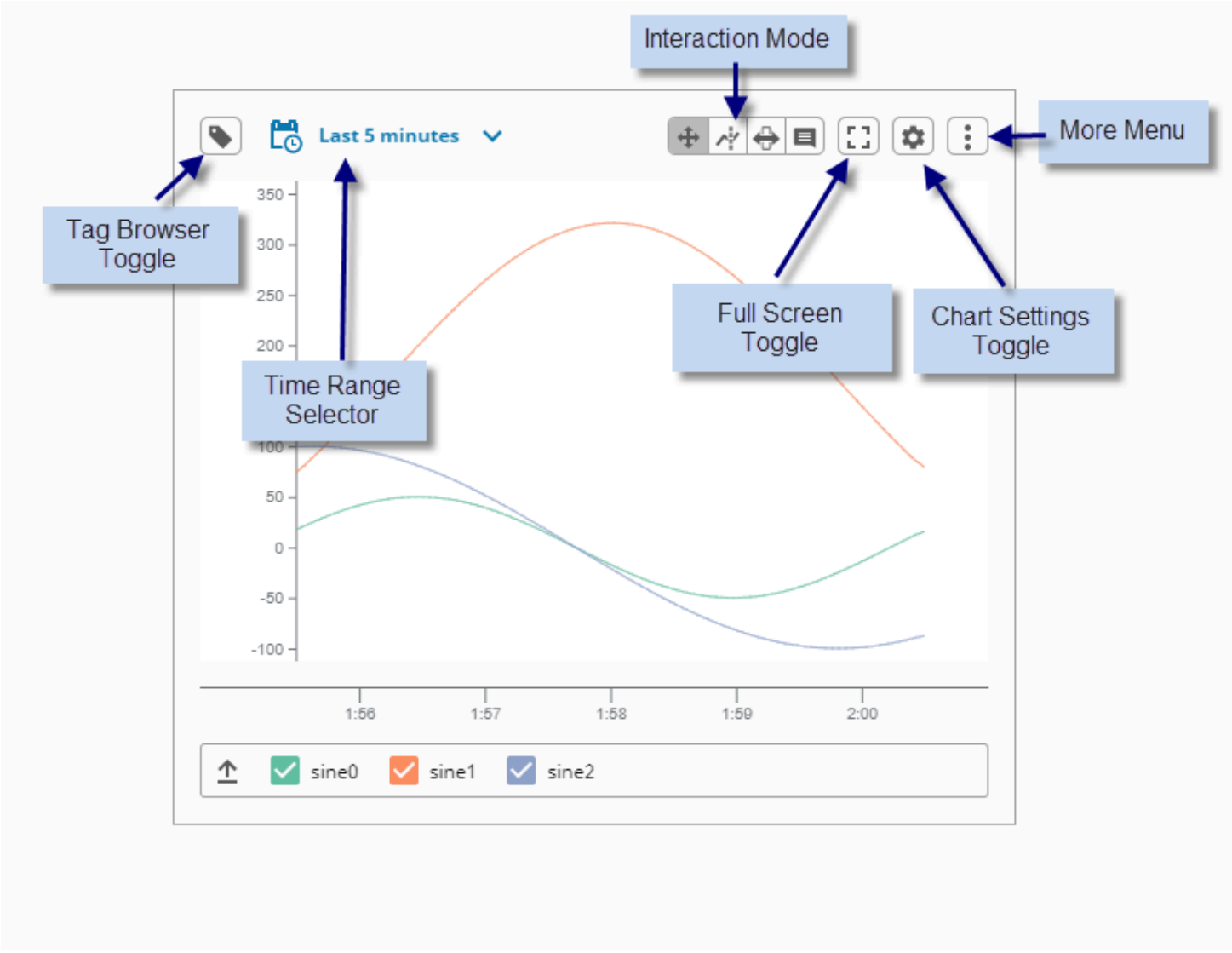
The Pen Data display supports numeric locale formatting. Numeric values are automatically formatted based on the Session locale and the **axes.dataFormat** property.

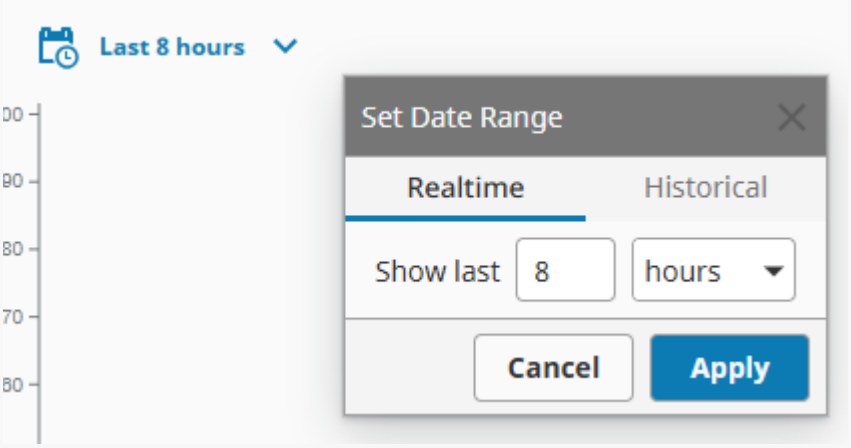
	Pen Name	Current Value	Minimum	Maximum	Average	Axis	Plot	X Trace
☑	sensor 2/reading	56,34	-39,32	158,85	53,89	Default	0	98,09
☑	sensor 3/reading	47,71	-60,08	152,45	47,37	Default	0	48,59
☑	ambienthum	102,65	-8,79	200,53	99,81	Default	0	102,69

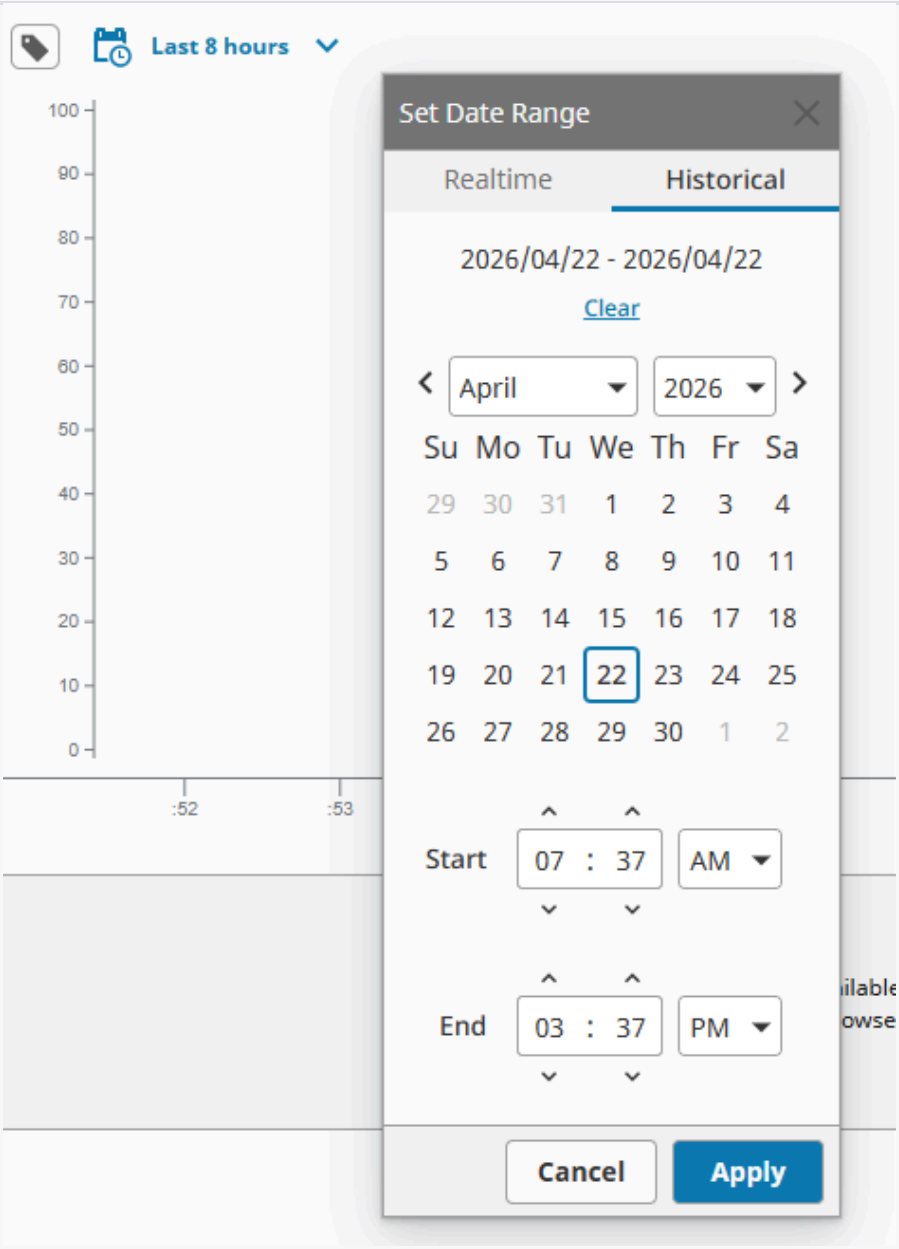
Chart Settings

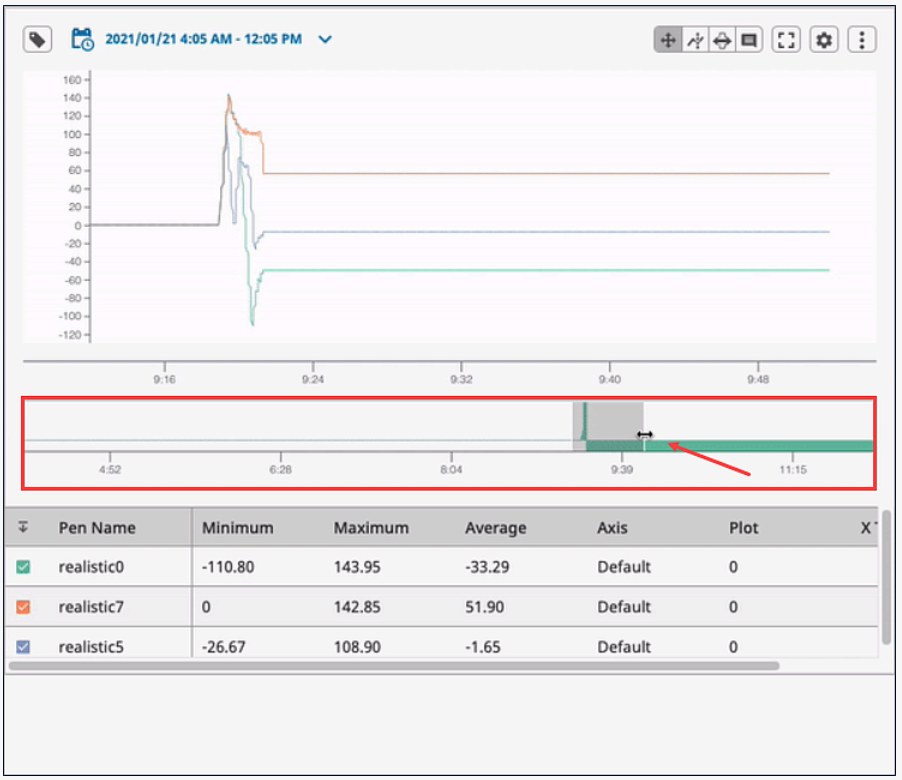
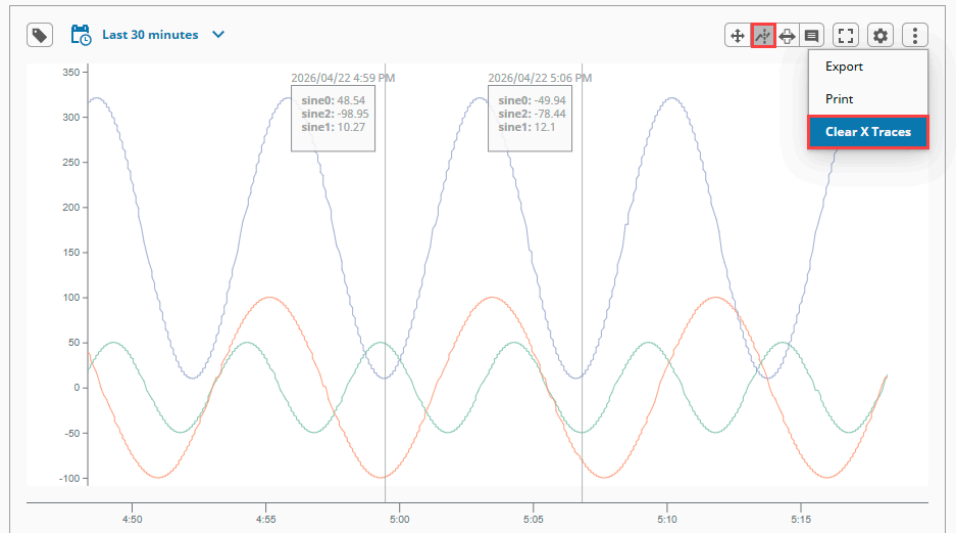
This panel allows users to add new objects to the chart, such as axes and plots.

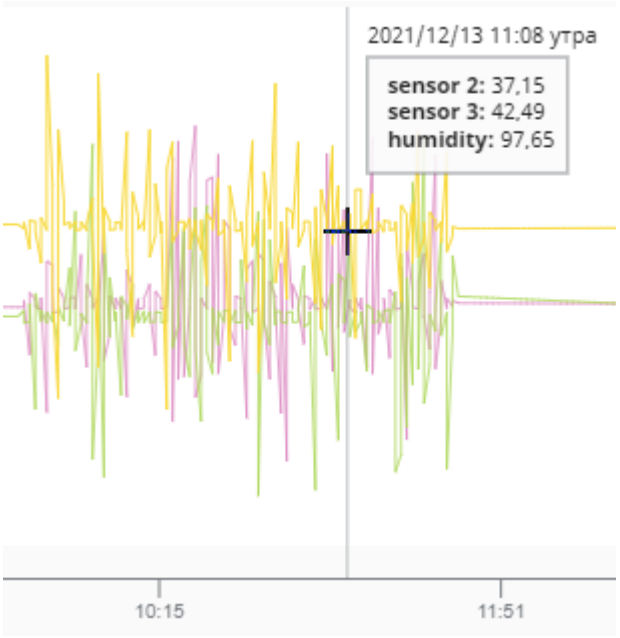
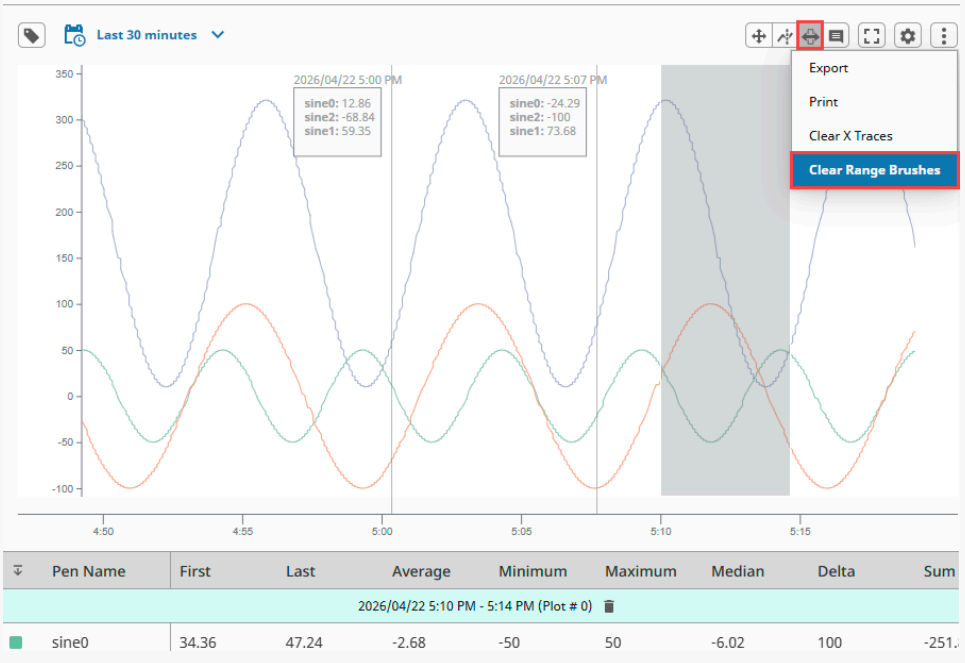
User Interaction

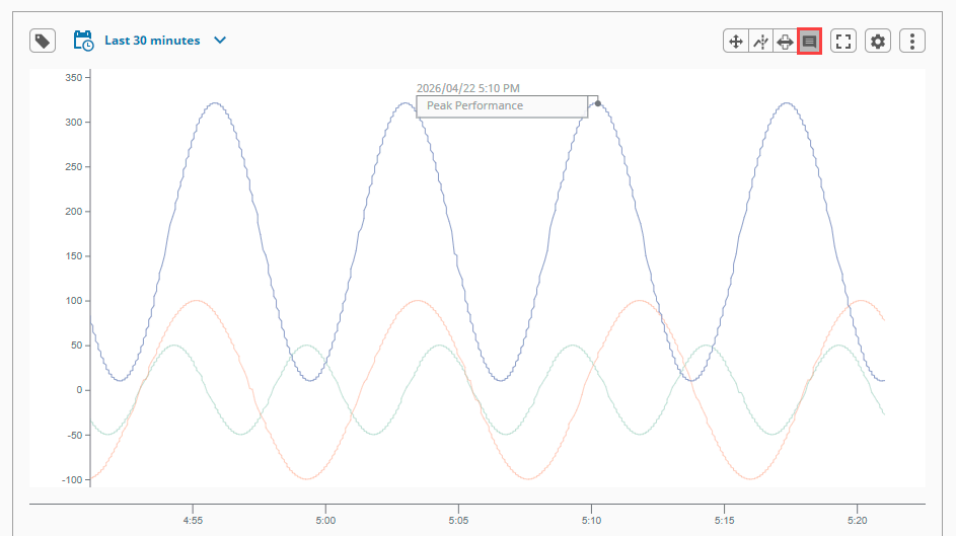


Interaction	Icon	Description
Browse Tags		Toggles the Tag Browser panel.
Date Range Selector		Allows you to set the range on the chart. There are two modes: • **Realtime**: Shows the most recent chart data based on the given timeframe. Useful in cases where you want to display records from now to some amount of time in the past.  • **Historical**: Allows you to pick at start datetime and end datetime using a popup calendar.

Interaction	Icon	Description
		 When the chart is in Historical mode, an additional time axis appears. On this axis, you can narrow down what the chart is showing you to specific points of interest by clicking and

Interaction	Icon	Description
		dragging the mouse along the axis. 
Pan and Zoom		In this mode you can drag or swipe to pan forward and backward in time. On desktop device, clicking and dragging will pan across the chart. The mouse wheel will zoom. On mobile/touchscreen device, tapping and dragging will pan. The "pinch" and "spread" gestures will zoom.
Zoom reset		To reset the zoom to default, click the zoom reset icon.
X Trace		Tap to place a vertical line, which shows an interpolated value for each pen on the plot. To clear the X Trace values, select **Clear X Trace** from the More Menu. 

Interaction	Icon	Description
		The X Trace display now supports numeric locale formatting. Numeric values are automatically formatted based on the Session locale and the **xTrace.infoBox.dataFormat** property. 
Range Brush		Allows you to select a range of data on the chart. While a brush is active, the Pen Control Table will show aggregate summaries for the pens over the selected range. Multiple brushes will create multiple aggregation summaries. Individual selections can be removed by clicking the trashcan icon in the Pen Control Table, or by selecting **Clear Range Brushes** on the More Menu. 
Annotate		Click near a trend, line, or data point and you'll have the opportunity to add an annotation. The annotation is stored with the Historian

Interaction	Icon	Description
		system, on the sqlth_annotations table. Annotations can edited by clicking the **Edit** icon and deleted by clicking the **Delete** icon, which appear when hovering over the annotation. 
Full Screen		Puts the chart into full screen mode.
Settings		Opens the Chart settings panel, allowing users to modify various aspects of the chart from the Session. There are four tabs: Axes, Pens, Plots, and Columns. Click to see the **settings** options.
More Menu		Provides additional contextual options, depending on the current state of the chart. The button to open the More Menu only appears if there is at least one pen on the chart. Click to see the **More Menu** options.

Settings

Tab	Description
Axes	You can add or delete Axes here, or click the **Edit** icon to edit an existing axis. The options are the same on the Add Axis and Edit Axes screens.
Pens	You can add or delete pens here or click the **Edit** icon to edit an existing pen. The options are the same on the Add Pen and Edit Pen screens.
Plots	You can add or delete plots here or click the **Edit** icon to edit an existing plot. The options are the same on the Add Plot and Edit Plot screens.
Columns	The Columns Chart Settings tab has options for datapoints to display on the Pen Control Panel and datapoints to display on the Range Brush Table.

More Menu

Option	Description
Export	Takes the datapoints visible on the various plots, and exports them to a CSV.
Print	Opens print dialog box so user can print the chart.
Clear X Traces	Clears all X Traces on the chart. Only appears when there are X Traces on the chart.
Clear Range Brushes	Clears all range brushes on the chart. Only appears when there are brush selections on the chart.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description
config	Configuration for the data feeding the chart. • **penNamePathDepth**: This value will set the depth of the tag path to inclu pen name. Using the Sample Quick Start project as an example, the **reali** has a parent folder of **realistic** and a grandparent folder of **sample-tags**. name in the Power Chart will display differently based on the value: ◦ `1`: realistic0 ◦ `3`: sample_tags/realistic/realistic1 You can also add new tags with path depth values that differ from existing needed for your project. Value is numeric. • **tagBrowserStartPath**: A path to a nested Tag History Provider structure f which browsing will start. The path is expected to contain key-value pairs separated by `:/` characters. Value is string. It is recommended to use the following syntax: `histprov:Test_DB:/sys:myGateway:/prov:Provider:/tag:Folder/New` ◦ **histprov**: The name of the Historian Provider. ◦ **sys**: The Gateway name. ◦ **prov**: The Tag Provider name. ◦ **tag**: A path to a node that has children. Typically should lead to either or the root node of a UDT instance. The tag component can be omitte will set the starting path for the Tag Browser panel to the root of a par driver. **Note**: The SQL Historian requires the `drv` key-value pair syntax. Addition `drv` syntax used in Ignition 8.1 will still work in 8.3. Refer to the **8.1 Powe** page for more details on this alternative syntax. • **mode**: The type of query that is being made against the data source. Opti realtime or historical. Value is string. • **refreshRate**: Duration (in milliseconds) that data will be queried for updat results. (realtime mode only). Value is numeric. • **pointCount**: Number of data points returned for the selected time range. Changing the pointCount property's value to -1 will retrieve pen data point stored in the database. In other words, this is the same as setting a Tag H binding's Query Mode to AsStored. While querying data with this mode, m value changes at the same timestamp will result in multiple rows, one row unique value. Value is numeric. • **startDate**: Start date for a historical data query (historical mode only). Val string.

Name	Description
	• **endDate**: End date for a historical data query (historical mode only). Valu string. • **dateFormat**: The date format displayed when in historical mode using a **MomentJS date string** (historical mode only). Value is string. • **timeFormat**: The time format displayed when in historical mode using usi **MomentJS time string** (historical mode only). Value is string. • **rangeSelectorPen**: The pen that will drive the data display of the range s (historical mode only). Value is string • **unitOfTime**: Time unit used for a realtime data query (realtime mode only is numeric. • **measureOfTime**: Time measurement used for a realtime data query. Opt seconds, minutes, hours, days, weeks, months, or years (realtime mode o Value is string. • **rangeStartDate**: Start date for the modified chart data range that the use selected, either with the range brush or by panning/zooming. Read-only. V date. • **rangeEndDate**: End date for the modified chart data range that the user h selected either with the range brush or by panning/zooming. Read-only. V date. • **responsiveDesignWidth**: A number (in pixels) that will be used as the sw width to the responsive design for the chart so it fits better on mobile devi mobile breakpoint is 750px, and is configurable. Value is numeric. • **visibility**: Settings to show/hide elements within the component interface. ◦ **showTagBrowser**: Flag representing the visible state of the Tag Brov Toggling this property will show or hide the Tag Browser. Value is boo ◦ **showDateRangeSelector**: Flag representing the visible state of the D Range Selector. Toggling this property will show or hide the Date Ran Selector. Value is boolean. ◦ **showPenControlDisplay**: Flag representing the visible state of the P Control display. Toggling this property will show or hide the Pen Cont display. Value is boolean. ◦ **buttons**: Settings to show/hide the buttons used in the interface. Clic more **buttons** properties. • **export**: Settings to control the format of data exported from the chart via t button's Export option.

Name	Description
	◦ **dateFormat**: The date format of the exported data. See **https://numeraljs.com** for formats. Value is string. ◦ **timeFormat**: The time format of the exported data. See **https://momentjs.com** for formats. Value is string
interaction	Configuration for the presentation of, and interaction with, chart data. • **mode**: Current user interaction mode of the chart. Options are panAndZo xTrace, rangeBrush, or annotation. Value is string. Click to see more **mod** properties. • **fullscreen**: Flag representing the full screen presentation mode of the cha is boolean. • **chartZoomLevel**: Read-only value that corresponds to the current zoom the chart. External changes to this value will not update the level of zoom displayed on the chart. Value is numeric. • **rangeZoomLevel**: Read-only value that corresponds to the current zoom the range brush when the chart is in Historical Mode. External changes to value will not update the level of zoom displayed on the chart. Value is nu
axes	Collection of predefined axes against which the data visualizations can be dra • **name**: The name of the axis. Value is string. • **position**: The side of the plot upon which the axis should be drawn. Optic left or right. Value is string. • **width**: The width of the axis. Value is numeric. • **color**: The color of the Y axis vertical bar. • **dataFormat**: A numeral.js data format for displaying the data displayed in control portion of the chart for this axis. See **https://numeraljs.com** for fo Value is string. • **range**: Configuration for the upper and lower limits of the axis. ◦ **auto**: If true, the minimum and maximum displaying values for the ax auto calculated. Auto range Y axes are now constrained to the upper limits of the pen data bound to said axis. Value is boolean. ◦ **min**: Minimum range value. If no value is provided, a minimum value calculated from the data bound to this axis. Value is numeric. ◦ **max**: Maximum range value. If no value is provided, a maximum valu calculated from the data bound to this axis. Value is numeric.

Name	Description
	• **label**: The label configuration for the Y axis. ◦ **text**: The text of the Y axis label. Value is string. ◦ **offset**: Offset the Y axis label from its default position. This enables y tune the label location, which may be necessary depending on the sc how much room the tick labels take up. Value may be positive or neg Default is 0. Value is numeric. ◦ **font**: Font configuration for the Y axis label. Font size and color optio font. ◦ **style**: Style for the Y axis label. Full menu of **style options** is availab can also specify a **style class**. • **tick**: Tick configuration. Click to see **tick** properties. • **grid**: Configuration for gridlines to display on this axis. ◦ **visible**: Visible state of the gridlines. Gridlines are shown only for axe connect directly to the chart. Any satellite axes will display their tick configurations instead of gridlines. Value is boolean. ◦ **color**: Color of the gridlines. ◦ **opacity**: Opacity of the gridlines. Value is numeric. ◦ **dashArray**: Dashed appearance of the gridlines. Value is numeric. ◦ **style**: Style for the gridlines. Full menu of **style options** is available. also specify a **style class**. • **style**: Style for the display. Full menu of **style options** is available. You ca specify a **style class**.
pens	Visual representation of each active item. • **name**: Name of the pen. Value is string. • **visible**: Whether or not the pen is visible on the chart. Value is boolean. • **enabled**: Availability of the pen on the chart and pen configuration panel. boolean. • **selectable**: Flag to allow the pen to be responsive to user selection. Valu boolean. • **axis**: Name of an axis in the axes array to plot against. If left blank, a defa will be created based on data values. Value is string. • **plot**: The plot to which this pen is bound. Value is numeric. • **display**: Configuration that drives the display of the pen. Click to see **disp** properties.

Name	Description
	• **data**: ◦ **source**: Source or tag path of the data behind the pen. Non-historica paths are converted to a historical format (e.g. `histprov:default:/tag:_Simulator_/Ramp/Ramp0`). Any updates t via the UI will write the converted path back to this property. This kee the data.source properties in a historical format while still allowing an non-historical tag path to add pen data to the chart display. Conversi supported for tags that have history enabled through the Tag Editor. V string. ◦ **aggregateMode**: Mode to use to group the data. Value is string. ▪ **default (MinMax):** Returns both the minimum and maximum val each interval. ▪ **Average:** The mean of the values in each interval. ▪ **MinMax:** Returns the minimum and maximum values. ▪ **LastValue:** The last recorded value in the interval. ▪ **SimpleAverage:** Arithmetic mean, no weighting. ▪ **Sum:** Total of all values in the interval. ▪ **Minimum:** Lowest value in the interval. ▪ **Maximum:** Highest value in the interval. ▪ **DurationOn:** Total time the tag was in the On state. ▪ **DurationOff:** Total time the tag was in the Off state. ▪ **CountOn:** Number of On state transitions or samples. ▪ **CountOff:** Number of Off state transitions or samples. ▪ **Count:** Number of samples in the interval. ▪ **Range:** Difference between maximum and minimum values. ▪ **Variance:** Statistical variance of the values. ▪ **StdDev:** Standard deviation of the values. ▪ **PctGood:** Percentage of samples considered good quality. ▪ **PctBad:** Percentage of samples considered bad quality.
plots	A plot represents a row containing one or more pens. • **relativeWeight**: Ratio between all plots. Value is numeric. • **color**: Background color of the plot.

Name	Description
	• **markers**: An array of markers that can be added to the plot to better visua data being displayed. ◦ **type**: Type of marker to add to the plot. Options are line or band. Valu string. ◦ **axis**: Name of the axis against which the marker should be drawn. Th be specified for the marker to be drawn. Value is string. ◦ **value**: Value where the line marker should be drawn. Value is numeri ◦ **display**: Configuration for the display of the marker. Click to see **disp** properties. • **style**: Style for the individual plot. Full menu of **style options** is available background, margin and padding, border, shape and miscellaneous. You specify a **style class**
dataColumns	Configuration for the data columns that can be shown in tabular displays throu the chart. • **penControl**: Configuration for the data columns that can display for pens ◦ **currentValue**: Show the "current value" column for the pen based on range. Value is boolean. ◦ **minimum**: Show the minimum column for the pen based on the time Value is boolean. ◦ **maximum**: Show the maximum column for the pen based on the time Value is boolean. ◦ **average**: Show the "average" column for the pen based on the time r Value is boolean. ◦ **axis**: Show the axis column for the pen. Value is boolean. ◦ **plot**: Show the plot column for the pen. Value is boolean. ◦ **xTrace**: Show the xTrace column for the pen based on the time range is boolean. • **rangeSelection**: Configuration for the data columns that can display for th brush. ◦ **first**: Show the first column for the pen based on the time range. Valu boolean. ◦ **last**: Show the last column for the pen based on the time range. Valu boolean.

Name	Description
	◦ **average**: Show the average column for the pen based on the time ra Value is boolean. ◦ **minimum**: Show the minimum column for the pen based on the time Value is boolean. ◦ **maximum**: Show the maximum column for the pen based on the tim Value is boolean. ◦ **median**: Show the median column for the pen based on the time ran is boolean. ◦ **delta**: Show the delta column for the pen based on the time range. V boolean. ◦ **sum**: Show the median column for the pen based on the time range. boolean. ◦ **ucl**: Show the UCL column for the pen based on the time range. Valu boolean. ◦ **lcl**: Show the LCL column for the pen based on the time range. Value boolean. ◦ **standardDeviation**: Show the standardDeviation column for the pen on the time range. Value is boolean.
title	Configuration for the title of the chart. • **visible**: Whether or not the title is visible. Value is boolean. • **text**: The text for the title. Value is string. • **font**: Title font configuration. Font size and color options for the font. • **style**: Style for the display. Full menu of **style options** is available. You c specify a **style class**.
timeAxis	Configuration for the time axis (X axis) of the chart. • **visible**: Whether or not the time axis is visible. Value is boolean. • **tickCount**: The number of ticks on the axis. Setting the tickCount to 0 will automatically scale the chart's X Axis based on the zoom level, panning, scrolling. Value is numeric. • **height**: The height of the time axis. Value is numeric. • **color**: The color of the axis. • **tick**: Tick configuration. Click to see **tick** properties.

Name	Description
	• **grid**: Configuration for gridlines to display on this axis. ◦ **visible**: Visible state of the gridlines. Gridlines are shown only for axe connect directly to the chart. Any satellite axes will display their tick configurations instead of gridlines. Value is boolean. ◦ **color**: Color of the gridlines. ◦ **opacity**: Opacity of the gridlines. Value is numeric. ◦ **dashArray**: Dashed appearance of the gridlines. Value is numeric. ◦ **style**: Style for the gridlines. Full menu of **style options** is available. also specify a **style class**. • **style**: Sets a style for this timeAxis. Full menu of **style options** is availabl text, background, margin and padding, border, shape and miscellaneous. also specify a **style class**.
legend	Configuration for the display of the legend for the chart. • **visible**: Whether or not the legend is visible. Value is boolean.
style	Sets a style for this chart. Full menu of **style options** is available for text, back margin and padding, border, shape and miscellaneous. You can also specify a **class**.

config.visibility.buttons Properties

Name	Description	Property Type
showTagBrowserButton	Flag representing the visible state of the Open Tag Browser and Close Tag Browser buttons.	value: boolean
showPanZoomButton	Flag representing the visible state of the Pan/Zoom toggle button.	value: boolean
showXTraceButton	Flag representing the visible state of the X Trace toggle button.	value: boolean
showRangeBrushButton	Flag representing the visible state of the Range Brush toggle button.	value: boolean
showAnnotationButton	Flag representing the visible state of the Annotation toggle button.	value: boolean
showFullscreenButton	Flag representing the visible state of the Fullscreen toggle button.	value: boolean
showSettingsButton	Flag representing the visible state of the Settings toggle button.	value: boolean
showMoreButton	Flag representing the visible state of the Show More toggle button.	value: boolean

interaction.mode Properties

Name	Description	Property Type
panAndZoom	Configuration settings for the chart in panAndZoom mode. • **freeRange**: When enabled, this setting allows panning and zooming to dictate the time range used for the chart display. Value is boolean.	object
xTrace	Configuration settings for the chart in xTrace mode. • **values**: An array of read-only timestamp values representing the visible x-trace positions. • **infoBox**: Configuration to build the box portion of the x-trace display. ◦ **visible**: Whether or not the box is visible. Value is boolean. ◦ **showTime**: Whether or not the time value above the box is visible. Value is boolean. ◦ **width**: The width of the box. Value is numeric. ◦ **dataFormat**: A numeral.js data format for displaying the data for this axis. See **https://numeraljs.com** for formats. Setting the dataFormat property to an empty string will result in no formatting being applied to the value. Useful in cases where a binding is returning the data in a preformatted state. Value is string. ◦ **dateFormat**: The date format displayed when in historical mode using a MomentJS date string **(https://momentjs.com)**. Value is string. ◦ **timeFormat**: The time format displayed when in historical mode using using a MomentJS time string **(https://momentjs.com)**. Value is string. ◦ **stroke**: A configuration object describing the properties that will be applied to the stroke of the box. Click to see **stroke** properties. ◦ **fill**: The fill configuration. Click to see **fill** properties.	object

Name	Description	Property Type
	◦ **style**: Style for the box. Full menu of **style options** is available. You can also specify a **style class**. • **line**: Configuration to build the vertical line portion of the x-trace display. ◦ **visible**: Whether or not the line is visible. Value is boolean. ◦ **color**: The color to apply to the line stroke, if applicable. ◦ **width**: The width to apply to the line stroke, if applicable. Value is numeric. ◦ **opacity**: The opacity to apply to the line stroke, if applicable. Value is numeric. ◦ **dashArray**: The spacing to apply between dashes of the line stroke, if applicable. Value is numeric. ◦ **style**: Style for the box. Full menu of **style options** is available. You can also specify a **style class**.	
rangeBrush	Configuration settings for the chart in rangeBrush mode. • **values**: An array of config objects to build each range brush. ◦ **start**: The start timestamp position. Value is string. ◦ **end**: The end timestamp position. Value is string. • **active**: Configuration to build the active range brush display. ◦ **color**: The color of the active range brush. ◦ **opacity**: The opacity of the active range brush. Value is numeric. ◦ **style**: Style settings for the active range brush. Full menu of **style options** is available. You can also specify a **style class**. • **inactive**: Configuration to build the inactive range brush displays. • **color**: The color of the inactive range brush. • **opacity**: The opacity of the inactive range brush. Value is numeric.	object

Name	Description	Property Type
	• **style**: Style settings for the inactive range brush. Full menu of **style options** is available. You can also specify a **style class**.	
annotation	Configuration settings for the chart in annotation mode. • **infoBox**: Configuration to build the box portion of the annotation display. ◦ **visible**: Whether or not the box is visible. Value is boolean. ◦ **showTime**: Whether or not the time value above the box is visible. Value is boolean. ◦ **width**: The width of the box. Value is numeric. ◦ **dateFormat**: The date format displayed when in historical mode using a MomentJS date string **(https://momentjs.com)**. Value is string. ◦ **timeFormat**: The time format displayed when in historical mode using using a MomentJS time string **(https://momentjs.com)**. Value is string. ◦ **stroke**: A configuration object describing the properties that will be applied to the stroke of the box. Click to see **stroke** properties. ◦ **fill**: The fill configuration. Click to see **fill** properties. ◦ **font**: A configuration object describing the properties that will be applied to the font. Click to see **font** properties. ◦ **style**: Style for the box. Full menu of **style options** is available. You can also specify a **style class**. • **line**: Configuration to build the connecting line portion of the annotation display. ◦ **visible** Whether or not the line is visible. Value is boolean. ◦ **color**: The color to apply to the line stroke, if applicable. ◦ **width**: The width to apply to the line stroke, if applicable. Value is numeric.	object

Name	Description	Property Type
	◦ **opacity**: The opacity to apply to the line stroke, if applicable. Value is numeric. ◦ **dashArray**: The spacing to apply between dashes of the line stroke, if applicable. Value is numeric. ◦ **style**: Style for the box. Full menu of **style options** is available. You can also specify a **style class**. • **dot**: Configuration to build the dot portion of the annotation display. ◦ **visible**: Whether or not the dot portion is visible. Value is boolean. ◦ **color**: The color to apply to the dot. ◦ **radius**: The radius of the dot. Value is numeric. ◦ **opacity**: The opacity of the dot. Value is numeric. ◦ **style**: The style settings for the dot. Full menu of **style options** is available. You can also specify a **style class**.	

infoBox.stroke Properties

Both `interaction.mode.xTrace` and `interaction.mode.annotation` have the following infoBox.stroke properties.

Name	Description	Property Type
color	The color to apply to the line stroke, if applicable.	color
width	The width to apply to the line stroke, if applicable.	value: numeric
opacity	The opacity to apply to the line stroke, if applicable.	value: numeric
dashArray	The spacing to apply between dashes of the line stroke, if applicable.	value: numeric

infoBox.fill Properties

Both `interaction.mode.xTrace` and `interaction.mode.annotation` have the following infoBox.fill properties.

Name	Description	Property Type
color	The color to apply to the trend fill, if applicable.	color
opacity	The opacity to apply to the trend fill, if applicable.	value: numeric

interaction.mode.annotation.infoBox.font Properties

Name	Description	Property Type
color	The text color of the annotation label and datetime text.	color
size	The font size of the annotation label and datetime text.	value: numeric
style	Custom CSS styles to apply to the annotation text. Any style that applies to an SVG line element can be used. See also **style options**.	object

axes.tick Properties

Name	Description	Property Type
color	The color of the ticks.	color
count	Number of ticks to display on the Y axis. Default value is Auto. When the tick count is Auto, Y axis ticks will be added/removed based on the height of the chart, and the tick spacing is adjusted around whole numbers.	value: numeric or string
label	Tick label configuration. • **font**: Label font configuration. Font size and color options for the font. • **format**: Sets the numeric format for the tick label. The value must be a valid D3 Format value. Default value is Auto. When the tick label format is Auto, the D3 format ~f is used. This creates a fixed point that will trim any insignificant trailing zeroes. Value is string. • **style**: Style for the tick label. Full menu of **style options** is available. You can also specify a **style class**.	object
style	Style for the tick. Full menu of **style options** is available. You can also specify a **style class**.	object

pens.display Properties

Each of the following four `display` properties have the same set of `styles` **properties**.

Name	Description	Property Type
type	The type of chart to be built. Options are line, area, bar, or scatter.	value: string
interpolation	Type of curve that should be used to draw the line portion of the chart. Options are: curveBasis, curveBasisOpen,curveCardinal, curveCardinalOpen, curveCatmullRom, curveCatmullRomOpen, curveLinear, curveMonotoneX, curveMonotoneY, curveNatural, curveStep, curveStepAfter, or curveStepBefore. More information on the interpolation methods above can be found in D3's documentation.	value: string
breakLine	If true, the line will be broken on either side of bad/missing data values. If false, bad/missing data values are removed and the adjoining points are connected.	value: boolean

Name	Description	Property Type
styles	Settings for the display when it is normal, highlighted, selected, or muted. • **normal**: An object providing style configuration for the “normal” state (no user interaction) of a column, or data entry, in a trend. Any color values specified here will override values set in the colorScheme or colors properties. • **highlighted**: An object providing style configuration for the “highlighted” state (mouse hover) of a column, or data entry, in a trend. Any color values specified here will override values set in the colorScheme or colors properties. The highlighted property uses the same configuration properties as the 'normal' property above. • **selected**: An object providing style configuration for the “selected” state (mouse click) of a column, or data entry, in a trend. Any color values specified here will override values set in the colorScheme or colors properties. • **muted**: An object providing style configuration for the “muted” state (non selected) of a column, or data entry, in a trend. Any color values specified here will override values set in the colorScheme or colors properties.	object

pens.display.styles Properties

Name	Description	Property Type
stroke	A configuration object describing the properties that will be applied to the stroke of the trend type being displayed (if applicable). The line , and area trend types will have these styles applied to them. • **color**: The color to apply to the line stroke, if applicable. • **dashArray**: The spacing to apply between dashes of the line stroke, if applicable. Value is numeric. • **opacity**: The opacity to apply to the line stroke, if applicable. Value is numeric. • **width**: The width to apply to the line stroke, if applicable. Value is numeric.	object
fill	The fill configuration. • **color**: The color to apply to the trend fill, if applicable. • **opacity**: The opacity to apply to the trend fill, if applicable. Value is numeric.	object

plots.markers.display Properties

Name	Description	Property Type
color	Color of the marker.	color
width	Width of the line.	value: numeric
opacity	Opacity of the marker	value: numeric
dashArray	Dashed appearance of the marker.	value: numeric
label	Configuration for the label of the marker. • **text**: Text for the label. Value is string. • **position**: The position of the label relative to the line. Options are right or left. Value is string. • **font**: Label font configuration. Font size and color options for the font. • **style**: Style for the label. Full menu of **style options** is available. You can also specify a **style class**.	object
style	Style for the display. Full menu of **style options** is available. You can also specify a **style class**.	object

timeAxis.tick Properties

Name	Description	Property Type
color	The color of the ticks.	color
label	Tick label configuration • **angled**: Whether or not the tick labels are angled. Value is boolean. • **format**: Date/time format displayed by each tick using a MomentJS data string **(https://momentjs.com)**. Value is string. • **font**: Label font configuration. Font size and color options for the font. • **style**: Style for the tick label. Full menu of **style options** is available. You can also specify a **style class**.	object
style	Style for the tick. Full menu of **style options** is available. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

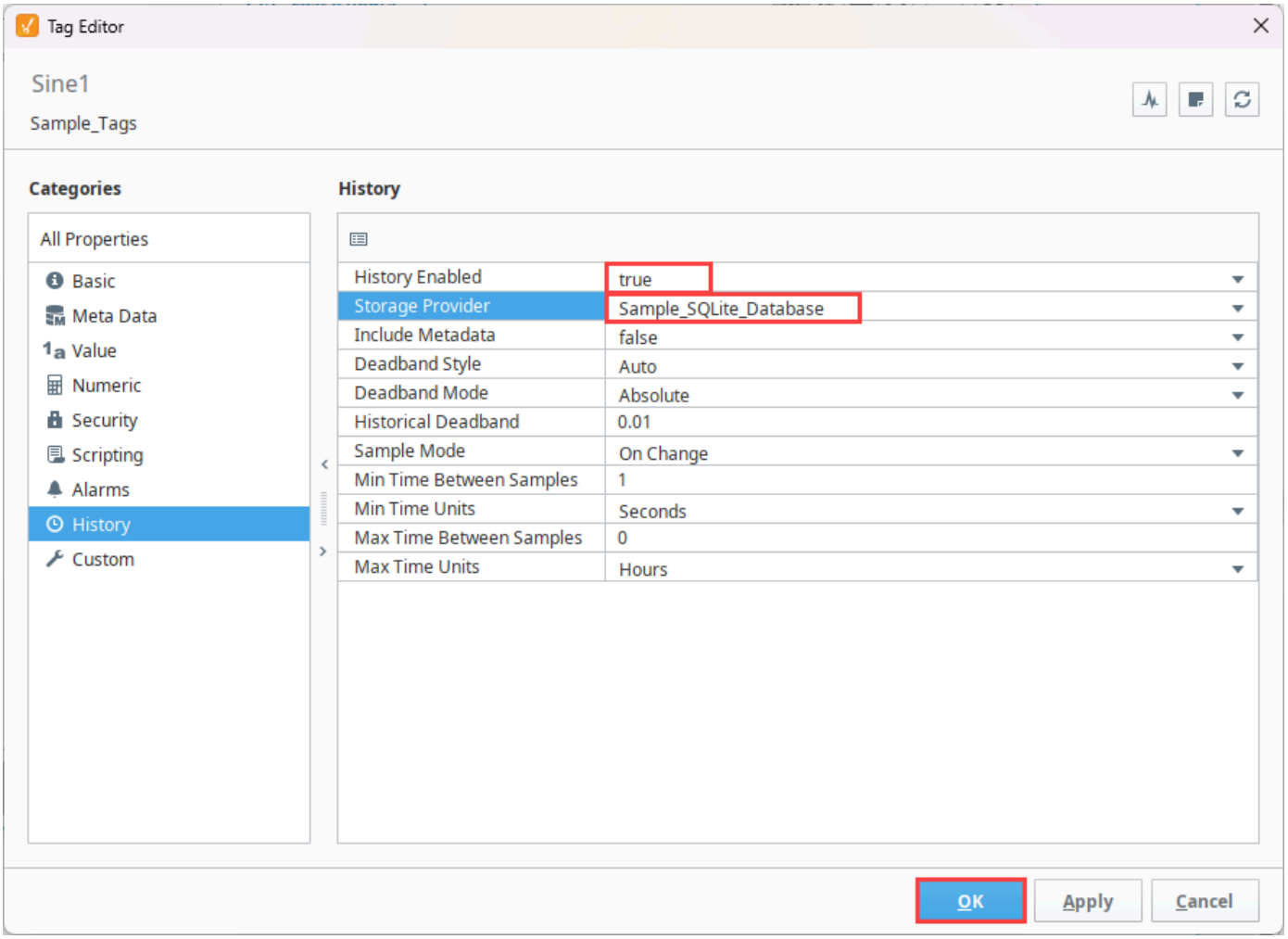
Example

For this example, we will be configuring history on three tags and adding them to the Power Chart. These tags are available from the Gateway Quick Start project, and the Historian will be the Sample_SQLite_Database.

1. Double-click the Sine0 tag from the Tag Browser to open the Tag Editor.

2. Access the History tab to set History Enabled to **true** and select **Sample_SQLite_Database** for the Storage Provider.

3. Select **OK**.



4. Repeat these steps for the Sine1 and Sine2 tags.

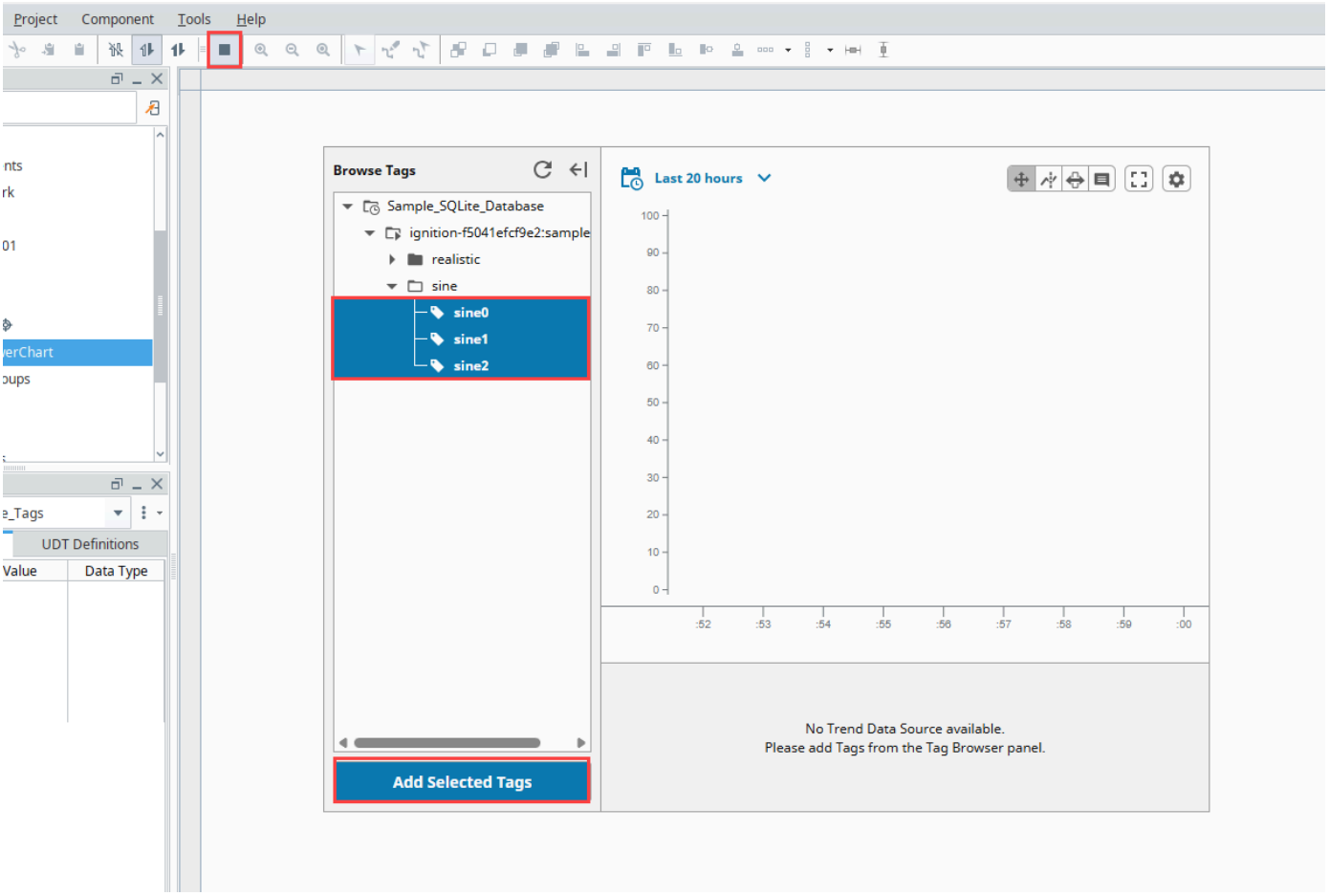
5. Drag and drop a Power Chart onto a view.

6. Place your Designer in Preview Mode.

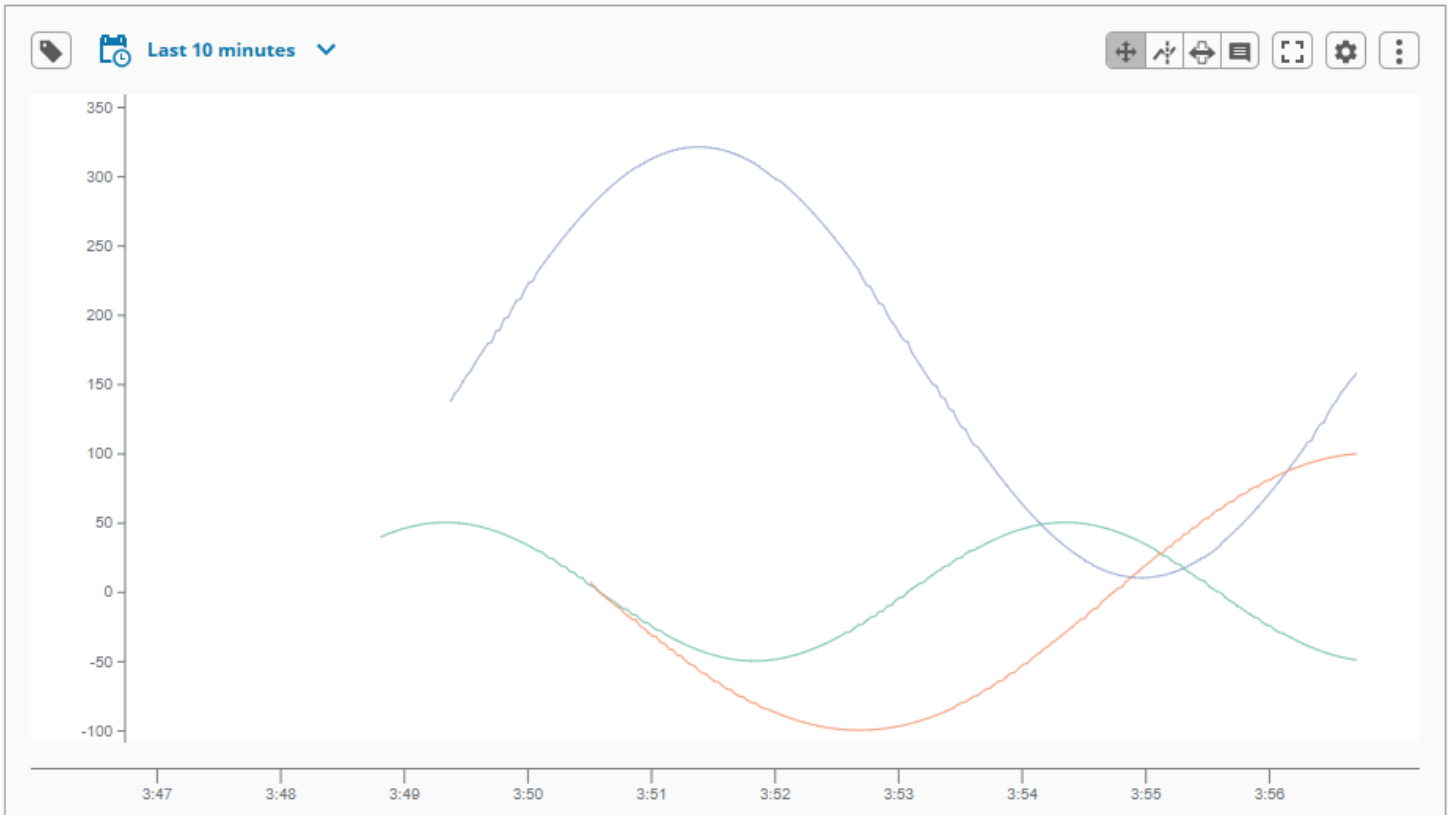
7. Expand the Browse Tags panel.

8. Expand the Historian Provider dropdown until the Sine tags are displayed.

9. Select the Sine0, Sine1, and Sine2 tags (using Crtl + click) and click **Add Selected Tags**.

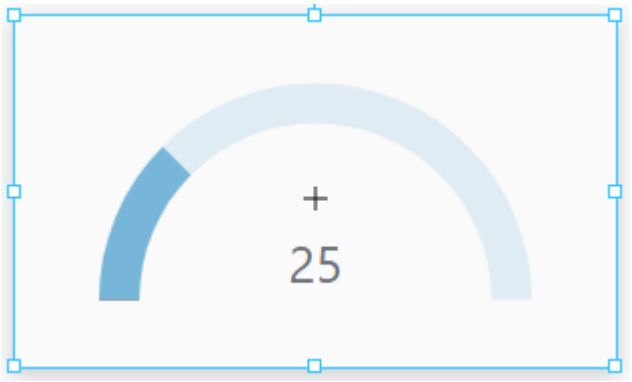


The Sine0, Sine1, and Sine2 tag data is now displayed on the Power Chart trend area. You can adjust the Date Range settings, pen controls, and chart settings from here to view the data as desired.



Pen Name	Current Value	Minimum	Maximum	Average	Axis	Plot	X Trace
sine0	-49.22	-50	50	2.13	Default	0	
sine2	99.44	-100	99.44	-21.98	Default	0	
sine1	157.75	10	321	165.93	Default	0	

Perspective - Simple Gauge



Component Palette Icon: 

The Simple Gauge component in Perspective provides a way to show realtime values in a range as they change. This gauge is a less complicated version of the **Gauge** component. It has just one axis, is easy to configure and is customizable in its appearance.

The Simple Gauge component has three pre-configured **variants**:

- Half Circle - Default layout with a half-circle gauge.
- 3/4 Circle - Layout with a 3/4 circle gauge.
- Full Axis - Layout with a full axis gauge.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

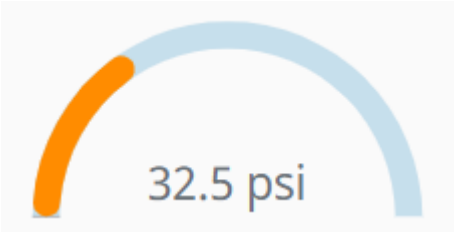
Name	Description	Property Type
value	Numeric value for the gauge to display. Default is 0.	value: numeric
minValue	Minimum gauge value for this gauge. Default is 0.	value: numeric
maxValue	Maximum gauge value for this gauge. Default is 100.	value: numeric
startAngle	Radial position for the start of the gauge's arc. Default is 180.	value: numeric
endAngle	Radial position for the end of the gauge's arc. Default is 360.	value: numeric
arc	The arc is a radial band that displays the gauge's value. • **width**: Width of the line (in pixels) that represents the arc. Default is 20. Value is numeric. • **color**: Color of the arc line showing the gauge's value. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. • **cornerRadius**: Amount to round the edges of the arc. Default is 0. Value is numeric.	object
arcBackground	Background or 'track' for the gauge arc. Shows shape and total potential value behind the arc. • **color**: Color of the arc background. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. • **opacity**: Opacity of the arc. 0 is fully transparent, 1 is fully opaque. Value is numeric.	value: numeric
label	The label for the gauge is displayed as text, with optional units.	object

Name	Description	Property Type
	• **visible**: Indicates whether or not the label is visible. Default is true (visible). Value is boolean. • **size**: Font size to display label text. Default is 25. Value is numeric. • **offsetX**: Offset position on the x axis, relative to the middle of the gauge. Value is numeric. • **offsetY**: Offset position on the y axis, relative to the middle of the gauge. Value is numeric. • **color**: Color of the label. • **units**: Any unit information to append to the value on the label. Value is string. • **maxDecimal**: Maximum number of digits after decimal to display in the label. If null, full value will display. Value is numeric.	
animate	Whether needle should be animated in a sweeping motion when value changes. Default is false.	value: boolean
style	Sets a style for this gauge. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

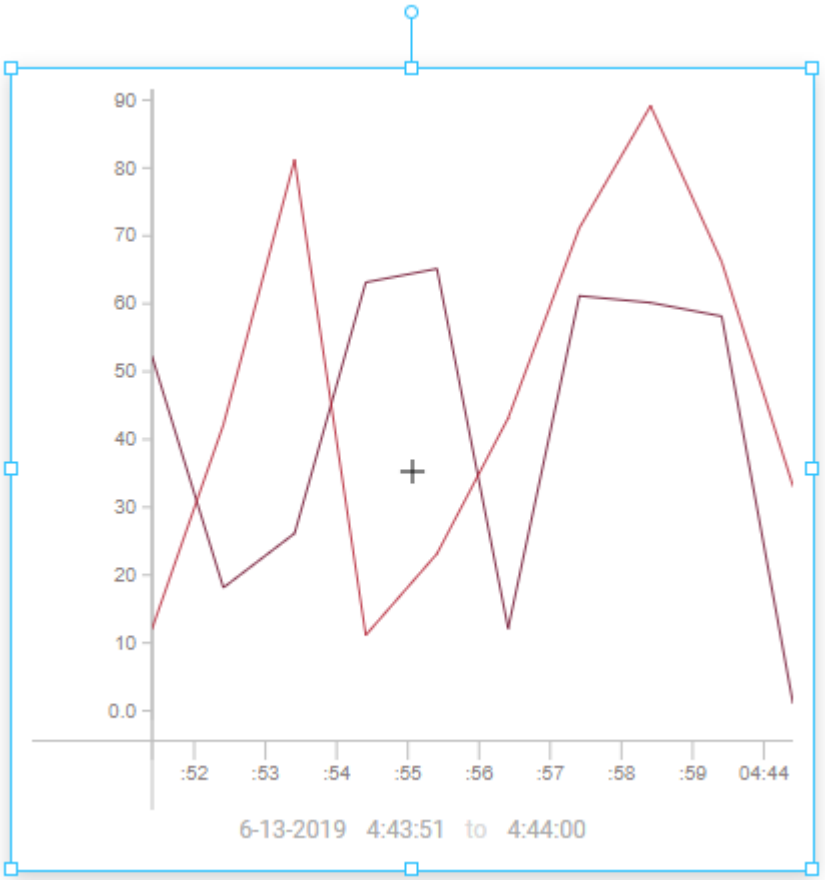
The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example



Property	Value
Value	32.4567
props.arc.width	15
props.arc.cornerRadius	25
props.arc.color	#FF8C00
props.arcBackground.opacity	0.4
props.label.units	psi
props.label.maxDecimal	1

Perspective - Time Series Chart



Component Palette



The Time Series component provides an efficient way to visualize data from a variety of different data sources as chart data. Time series chart requires that the X axis of the chart represents time and the Y axis represents values. The Time Series Chart includes the following features:

- Zoom or pan in and out via mouse wheel interaction.
- X-Trace display showing data at the hovered time position.
- Multiple chart display types (Area, Bar, Line, and Scatter).
- Multiple Y axes with the ability to align to the left or right side of the chart.
- Multiple plots as well as multiple trends per plot.
- Baselines and markers.
- Custom axes.
- Time range showing the overall range of the data being displayed in the chart
- Simple display customization for the axes, different trend display types, baselines, and markers.
- Label and Title properties have their own dedicate styling properties, such as color and size.

By default, the charts contain example data, but typically a **tag history binding** or **named query binding** will be used to feed data to the charts.

The **Chart Range Selector** provides a complement to this chart. The Time Series Chart and Chart Range Selector components are most powerful when paired together.

The Time Series Chart component has four pre-configured **variants**:

- Line chart - Default layout with appearance of a line chart.
- Area chart - Layout set up as an area chart.
- Bar chart - Layout set up as a bar chart.
- Scatter chart - Layout set up as a scatter chart.

User Interaction

Interaction	Description
Zoom	The user can zoom in and out on the Time Series chart. When zoomed to any level past its base time range, the Time Series chart will display a **zoom reset** icon in the upper right corner. Click on the icon to return the chart to its base range. Note that the range will not refresh while zoomed in. When pen data sources change, the zoom level will be reset to its base time range.
Pan	The user can pan across the Time Series chart. When panned past its base time range, the Time Series chart will display a **pan reset** icon in the upper right corner. Click on the icon to return the chart to its base range. Note that the range will not refresh while panning.
Pinch Zoom	On a mobile device, the user can pinch-zoom the Time Series chart. Zooms must originate from within the boundaries of the displaying chart data.
Tracker Position	On a mobile device, t he user can move the tracker position on the Time Series chart via touch.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
enablePanZoom	Allows the chart to be panned and zoomed. The chart cannot be zoomed out past its base range.	value: boolean
autoGenerateSeriesNames	If true, auto generates series names if no name is provided. This helps prevent the series name from resetting to the default value when the series name is provided by a binding.	value: boolean
series	A list of series entries used as the base data source for the chart display. Each series will be a new line drawn on the chart. At a minimum, the Time Series component requires at least one complete series entry to display data. With no other configuration provided, an initial display will be created from the series data (required). • **name**: The name of the series. A name must be provided or one will be added. This will also be used as the label of the auto-generated Y axis in the chart (required). Value is string. • **data**: Data can be an object containing a time entry and value entries (all must be numbers) (required). Each value entry must be labeled with the column name to which it corresponds. Data can also be an array containing value entries (all must be numbers). Each value entry consists of a timestamp (which must be the first value) and one or more values that were captured at that time. Finally, data can also be in the form of a DataSet, for example, the data property can be bound to a Tag History binding to display either realtime data, or historical data (via start and end dates). **Note**: When using a dataset, the Time column should be the first column. Value can be object or array	array

Name	Description	Property Type
plots	A list of plots (subplots) for the chart. At least one entry is required. Plot entries contain properties that allow much finer control over the way that the series data can be displayed. Each plot will be represented by its own row in the component (plots always stretch to fill the width of the chart). • **relativeWeight**: Relative weight sets the amount of height space plots take up in a Time Series Chart when multiple plots are included. By default, plots are drawn using equal height with a value of 1. Customizing the value adjusts the plot height to the corresponding ratio of available space in the chart. Value is numeric. • **trends**: If specified, a trend will become the display mechanism for the plot in which it resides. It describes the way that the data should look, and provides different display types as opposed to the line display type that is provided when building from series data. Multiple trends will be built on top of each other in the same plot. An axis will be generated for each trend , unless they share a common axis (either custom created, or generated from series data). **Note**: When making changes to a setting under trends, an axis must also be defined under plots.axes ◦ **visible**: If specified, this is the visible state of the trend. If not visible, the trend data will be hidden, but the time range of the trend will still be represented in the overall time range of the plot. Value is boolean. ◦ **type**: The type of chart to create (required). Options are area, bar, line, and scatter. Value is string. ◦ **series**: The series used to feed data to this trend (required). Value is string. ◦ **interpolation**: The type of curve that should be used to draw the line portion of the chart.	array

Name	Description	Property Type
	Options are: curveBasis, curveBasisOpen, curveCardinal, curveCardinalOpen, curveCatmullRom, curveCatmullRomOpen, curveLinear, curveMonotoneX, curve MonotoneY, curveNatural, curveStep, curveStepAfter, curveStepBefore. More information on the interpolation methods above can be found in D3's documentation. Value is string ◦ **breakLine**: This property will be available when a trend of type **line** is being used. If true, the line will be broken on either side of missing data values. If false, missing data values are removed and the adjoining points are connected. Value is boolean. ◦ **stack**: This property will be available when a trend of type **area** or bar is being used. If true, the multiple columns of the chart will stack on top of each other. Value is boolean. ◦ **axis**: If specified, the name of an axis that is described in the axesproperty of the plot to which this trend belongs. This axis will be used as the Y axis for the trend. Value is string. ◦ **radius**: This property will be available when a trend of type **scatter** is being used. A number specifying the radius (in pixels) of the displaying points. Value is numeric. ◦ **columns**: If provided, only the columns in this list will be shown. Any style properties provided here will also override any existing style currently in place. Click to see **columns** properties. ◦ **baselines**: If specified, a line will be drawn on the trend based upon a given type of functionality. Click to see **baselines** properties.	

Name	Description	Property Type
	• **axes**: A list of axis entries for that can be used for the plot. These can be used to override the Y Axis that is generated based on data in the series to which each trend is bound. **Note**: When making changes to a setting under axes, a trend must also be defined under plots.trends. Click to see **axes** properties. • **markers**: Settings for the markers, a list of visual indicators that can be added to the plot (optional). These are meant to draw emphasis to the data. In this first release, the line marker will be the only available option. Click to see **markers** properties.	
title	Settings for the title of the chart. • **visible**: Indicates whether or not the title is visible. Default is false. Value is boolean. • **text**: Text for the title of the chart. Value is string. • **height**: The vertical space taken up by the title. Value is numeric. • **font**: Title font configuration. ◦ **color**: Color of the title text. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. Value is string. ◦ **size**: The font size, in pixels, of the title text. Value is numeric. • **style**: Sets a style for the title. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
timeAxis	This property provides settings for the X Axis. Note that multiple plots share the same axis. (required) • **visible**: The visible state of the the axis. Value is boolean	object

Name	Description	Property Type
	• **tickCount**: The number of ticks on the axis (as a multiple of 2, 5, or 10). Setting the tickCount to 0 will automatically scale the chart's X Axis based on the zoom level, panning, and scrolling. Value is numeric • **height**: The height of the axis. Value is numeric • **color**: The color of the axis. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. Value is string • **tick**: The configuration of the ticks on the axis. Click to see **tick** properties. • **grid**: Configuration for gridlines to display on this axis. ◦ **visible**: Visible state of the gridlines. Gridlines are shown only for axes that connect directly to the chart. Any satellite axes will display their tick configurations instead of gridlines. Value is boolean. ◦ **color**: Color of the gridlines. ◦ **opacity**: Opacity of the gridlines. Value is numeric. ◦ **dashArray**: Dashed appearance of the gridlines. Value is numeric. ◦ **style**: Style for the gridlines. Full menu of **style options** is available. You can also specify a **style class**. • **style**: A style object containing properties which are applied to the horizontal line of the axis. Any property that would apply to an SVG line element can be used here. See also **style options**.	
timeRange	An object describing the presentation of the time range display below the chart.	object

Name	Description	Property Type
	• **visible**: The visible state of the time range display as a whole. Value is boolean. • **dateFormat**: The date format of the range using a **MomentJS** date string. Value is string. • **timeFormat**: The time format of the range using a **MomentJS** date string. Value is string.	
xTrace	Configuration to build the x-trace display when hovering over the chart. The X Trace display now supports numeric locale formatting. Numeric values are automatically formatted based on the session locale and the dataFormat, dateFormat, and timeFormat properties. • **value**: A read only timestamp representing the current x-trace position. If there is no active x-trace position, this value will be an empty string. Value is string. • **visible**: The visible state of the x-trace display. Value is boolean. • **infoBox**: Configuration to build the box portion of the x-trace display. ◦ **visible**: The visible state of the box. Value is boolean. ◦ **showTime**: Whether to display the timestamp of the current X Trace value above the info box. Value is boolean. ◦ **width**: Width of the info box, in pixels. Value is numeric. ◦ **dateFormat**: The date format of the xtrace date/time display using a Momentjs date string (**https://momentjscom/**). Options are: [7-18-2019], [2019-7-18], [07-18-2019], [2019-07-18], [Jul 18th 19], [Jul 18th 2019], or none. Value is string.	object

Name	Description	Property Type
	◦ **timeFormat**: The time format of the xtrace date/time display using a MomentJS time string (**https://momentjs.com/**). Options are: 12 hour [8:41:06], 12 hour w/ period [8:41:06 AM], 24 hour [08:41:06], 24 hour w/milliseconds [08:41:06:269], Unix Millisecond Timestamp [1563464737269], Unix Timestamp [1563464737], or none. Value is string. ◦ **dataFormat**: A NumeralJS value used to format the data found at the current timestamp of the X Trace display. See **numeral.js** for a list of available data formats. Options are: number [1,000.12], integer [1,200], four decimal precision [1.1200], percent [10.12%], scientific [1.01E+03], accounting [$(1,000.12)], financial [(1,000.12)], currency [$1,000.12], currency (rounded) [$1,012], duration [24:01:00], abbreviation [1.2k], or ordinal [100th]. Value is string. ◦ **stroke**: A configuration object describing the properties that will be applied to the stroke of the box display. Click to see **stroke** properties. ◦ **fill**: A configuration object describing the properties that will be applied to the fill of the box display. Click to see **fill** properties. ◦ **font**: A configuration object describing the properties that will be applied to the font of text in the infoBox. Click to see **font** properties. ◦ **style**: Custom CSS styles to apply to the info box. Any style that applies to an SVG line element can be used. See also **style options**. • **line**: Configuration to build the vertical line portion of the x-trace display. Value is string.	

Name	Description	Property Type
	◦ **visible**: The visible state of the line. Value is boolean. ◦ **color**: The color of the line. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. Value is string. ◦ **width**: The width of the line, in pixels. Value is numeric. ◦ **opacity**: The opacity of the line, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. Value is numeric. ◦ **dashArray**: The dashed appearance (SVG dashed array) of the line. It's a list of space separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2". Value is numeric. ◦ **style**: Custom CSS styles to apply to the line. Any style that applies to an SVG line element can be used. See also **style options**	
legend	Configuration for the display and position of the legend for the Time Series Chart. • **visible**: The visible state of the legend. Default is false. Value is boolean. • **position**: The position of the legend. Options are top, right, bottom, or left. Value is a dropdown string. • **style**: A style object containing properties which are applied to the legend. See also **style options**. You can also specify a **style class**	object

Name	Description	Property Type
defaultStyles	An object providing style settings to the chart trends as a whole. • **colorScheme**: Specifies a Color Brewer color scheme to use on the series. See ColorBrewer2.org for available color schemes. Value is string. • **colors**: A list of colors to apply to the columns (in order) for each trend. If these values are provided, they will override the value provided for the colorScheme. Value is string. • **normal**: An object providing style configuration for the “normal” state (no user interaction) of a column, or data entry, in a trend. Any color values specified here will override values set in the colorScheme or colors properties. • **highlighted**: An object providing style configuration for the “highlighted” state (mouse hover) of a column, or data entry, in a trend. Any color values specified here will override values set in the colorScheme or colors properties. The highlighted property uses the same configuration properties as the 'normal' property above. • **selected**: An object providing style configuration for the “selected” state (mouse click) of a column, or data entry, in a trend. Any color values specified here will override values set in the colorScheme or colors properties. • **muted**: An object providing style configuration for the “muted” state (non selected) of a column, or data entry, in a trend. Any color values specified here will override values set in the colorScheme or colors properties. Click to see the related **stroke and fill** properties.	object

Name	Description	Property Type
style	Sets a style for this chart. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class** .	object

plots.trends.columns Properties

A column contains the following properties:

Name	Description	Property Type
key	This needs to match a column name from the series to which this trend is bound (required). Once in place, this will allow the style configuration provided here to override the styles provided in the defaultStyles property.	value: string
color	If provided, this value will override any previous color values for the column (both stroke and fill). Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	value: string
styles	If provided, the styles for the state values listed here will override any previous state values. • **normal**: Takes the same configuration options as the same named state value under the defaultStyles property. • **highlighted**: Takes the same configuration options as the same named state value under the defaultStyles property. • **selected**: Takes the same configuration options as the same named state value under the defaultStyles property. • **muted**: Takes the same configuration options as the same named state value under the defaultStyles property.	object

plots.trends.baselines Properties

Name	Description	Property Type
visible	Whether the baseline is visible or not.	value: boolean
function	The type of baseline that will be drawn (required). Options are min, max, avg, ucl (upper control limit), and lcl (lower control limit).	value: string
column	The column against which the baseline should be calculated. If not specified, the first column in the series to which the trend is bound will be used.	value: string
axis	The axis against which the baseline should be calculated. The trend must be bound to an axis for this to work correctly.	value: string
color	The color of the line. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	value: string
width	The width of the line, in pixels.	value: numeric
opacity	The opacity of the line, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric
dashArray	The dashed appearance (SVG dashed array) of the line. The pattern of dashes and gaps used to paint the stroke. It's a list of space separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".	value: numeric
label	The configuration used for the label drawn on the baseline. • **text**: The label text. Value is string.	object

Name	Description	Property Type
	• **position**: The position of the label relative to the baseline. Value is string. • **font**: The font style for the label. ◦ **color**: The color of the label text. Value is string. ◦ **size**: The font size, in pixels, of the label text. Value is numeric. • **style**: Custom CSS styles to apply to the baseline label. Any style that applies to an SVG text element can be used. See also **style options**.	

plots.axes Properties

An axis entry has the following properties:

Name	Description	Property Type
name	The name of the axis (required). Also used as the display label of the axis.	value: string
min	The minimum value of the axis (required). If no value is specified, auto range will be used. A minimum value will be calculated from the data bound to this axis.	value: numeric
max	The maximum value of the axis (required). The maximum range value of the axis. If no value is specified, auto range will be used. A maximum value will be calculated from the data bound to this axis	value: numeric
alignment	The side of the trend upon which the axis should be presented (required).	value: string
width	The width of the axis, in pixels (required).	value: numeric
label	The configuration of the Y axis label. • visible: Whether or not the label is visible. Value is boolean. • **text**: The text for the label. Value is string. • **offset**: Offset the Y axis label from its default position. This allows you to fine tune the label location, which may be necessary depending on the scale and how much room the tick labels take up. This may be positive or negative. Value is numeric. • **font**: The settings for the label's font. ◦ **color**: The color of the label text. Value is string. ◦ **size**: The font size, in pixels, of the label text. Value is numeric. • **style**: Custom CSS styles to apply to the Y axis label. Any style that applies to an SVG text element can be used. See also **style options**.	object

Name	Description	Property Type
tick	The configuration for the ticks drawn on the axis. • **color**: The color of the ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. Value is string. • **count**: Number of ticks to display on the Y axis. Default value is Auto. When the tick count is Auto, Y axis ticks will be added/removed based on the height of the chart, and the tick spacing is adjusted around whole numbers. Value is numeric or string. • **label**: The settings for the label on the tick. ◦ **format**: Sets the numeric format for the tick label. The value must be a valid **D3 Format** value. Default value is Auto. When the tick label format is Auto, the D3 format ~f is used. This creates a fixed point that will trim any insignificant trailing zeroes. The Y axis now supports numeric locale formatting. Numeric values are automatically formatted based on the session locale and the tick.label.format property. Value is string. ◦ **font**: The font style for the label. Click to see **font** properties. ◦ **style**: Custom CSS styles to apply to the baseline label. Any style that applies to an SVG text element can be used. See also **style options**. • **style**: Custom CSS styles to apply to the ticks. Any style that applies to an SVG text element can be used. See also **style options**.	object
grid	Configuration for gridlines to display on this axis. • **visible**: Visible state of the gridlines. Gridlines are shown only for axes that connect directly to the chart. Any satellite axes will display their tick configurations instead of gridlines. Value is boolean. • **color**: Color of the gridlines. • **opacity**: Opacity of the gridlines. Value is numeric.	object

Name	Description	Property Type
	• **dashArray**: Dashed appearance of the gridlines. Value is numeric. • **style**: Style for the gridlines. Full menu of **style options** is available. You can also specify a **style class**.	

plots.trends.axes.tick.label.font Properties

Name	Description	Property Type
color	The color of the label text.	value: string
size	The font size, in pixels, of the label text.	value: numeric

plots.markers Properties

Name	Description	Property Type
visible	Indicates whether or not the marker is visible.	Value is boolean
value	The numeric value represented by the marker.	Value is numeric
type	The type of marker. Currently the only option is line.	Value is string
axis	The axis against which the marker should be drawn (required).	Value is string
line	The configuration for the line portion of the marker. • **width**: The width of the marker line, in pixels. Value is numeric. • **color**: The color of the marker line. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. Value is string. • **opacity**: The opacity of the marker line, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. Value is numeric. • **dashArray**: The pattern of dashes and gaps (SVG dashed array) used to paint the marker line. It's a list of space separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5 3 2" is equivalent to "5 3 2 5 3 2". Value is numeric. • **label**: The configuration for marker's label. ◦ **text**: The marker label text. Value is string. ◦ **position**: The position of the label relative to the line. Value is string. ◦ **font**: The font settings for the label. Click to see **font** properties.	object

Name	Description	Property Type
	◦ **style**: Custom CSS styles to apply to the marker line. Any style that applies to an SVG line element can be used. See also **style options**. • **style**: Custom CSS styles to apply to the marker line. Any style that applies to an SVG line element can be used. See also **style options**.	

plots.markers.line.label.font Properties

Name	Description	Property Type
color	The color of the label text. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	value: string
size	The font size, in pixels, of the label text.	value: numeric
style	Custom CSS styles to apply to the marker label. Any style that applies to an SVG text element can be used. See also **style options**.	object

timeAxis.tick Properties

Name	Description	Property Type
color	The color of the ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. Value is string	
label	The configuration of the label drawn on the tick. • **angled**: If set to true, the tick labels will be angled rather than horizontal. Value is boolean • **format**: The date/time format displayed by each tick using a MomentjS date string (**https://momentjs.com/**). Default is “Auto”, where the property attempts to figure out the best format. For a listing of suggested formats, refer to **https://momentjs.com/docs/#/parsing/string-format/**. Value is string. • **font**: The settings for the label's font. ◦ **color**: Color of the label text. Value is string. ◦ **size**: The font size, in pixels, of the label text. Value is numeric. • **style**: Custom CSS styles to apply to the tick labels. Any style that applies to an SVG text element can be used. See also **style options**.	object

xTrace.infoBox.stroke Properties

Name	Description	Property Type
color	The color of the box stroke. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	value: string
width	The width of the box stroke, in pixels.	value: numeric
opacity	The opacity of the box stroke, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric
dashArray	The dashed appearance (SVG dashed array) of the box stroke. It's a list of space separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".	value: numeric

xTrace.infoBox.fill Properties

Name	Description	Property Type
color	The color of the box fill. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	value: string
opacity	The opacity of the box fill, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric

xTrace.infoBox.font Properties

Name	Description	Property Type
color	The text color of the info box label and datetime text.	color
size	The font size of the info box label and datetime text.	value: numeric
style	Custom CSS styles to apply to the info box text. Any style that applies to an SVG line element can be used. See also **style options**.	object

defaultStyles stroke and fill Properties

The defaultStyles normal, highlighted, selected, and muted properties all include the following stroke and fill properties.

Name	Description	Property Type
stroke	A configuration object describing the properties that will be applied to the stroke for the “selected” state (mouse hover) of a column, or data entry, in a trend. The line and area trend types will have these styles applied to them. • **opacity**: The opacity of the trend stroke, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. Value is numeric.	object
fill	A configuration object describing the properties that will be applied to the stroke for the “selected” state (mouse hover) of a column, or data entry, in a trend. The area, and box, and scatter trend types will have these styles applied to them. • **opacity**: The opacity of the trend stroke, ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. Value is numeric.	object

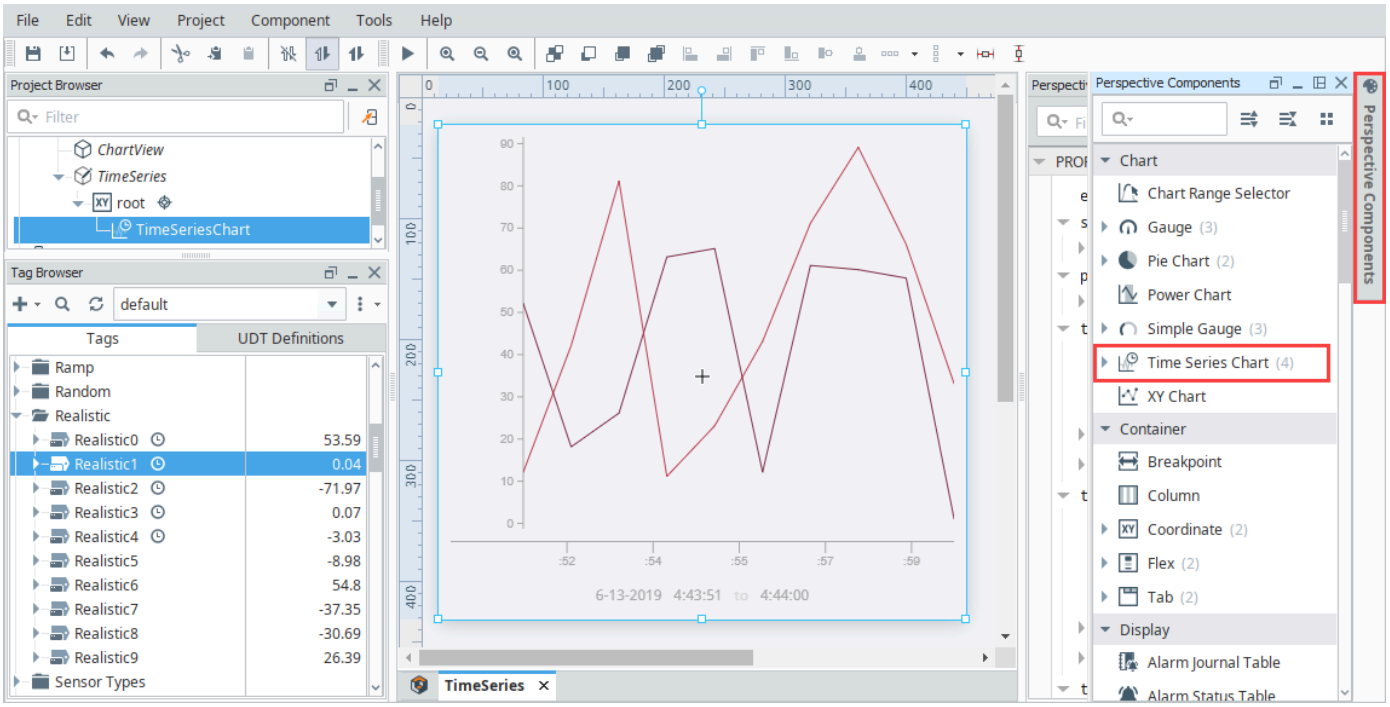
Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

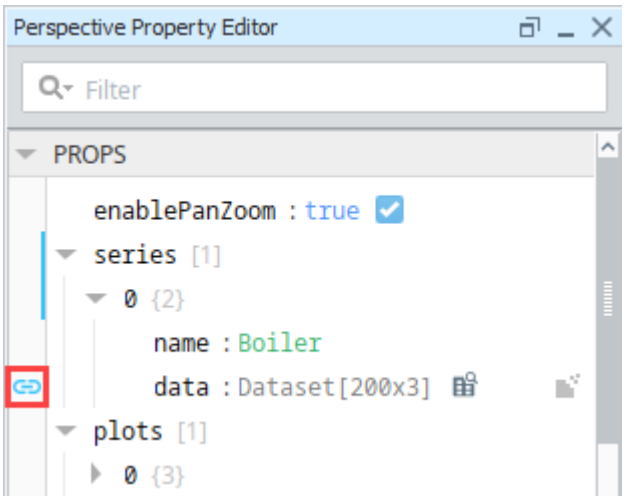
Examples

This example shows a Time Series Chart displaying Temperature and Pressure values for Tank 100.

1. Create a new view, and drag an **Time Series Chart** component on to your view.

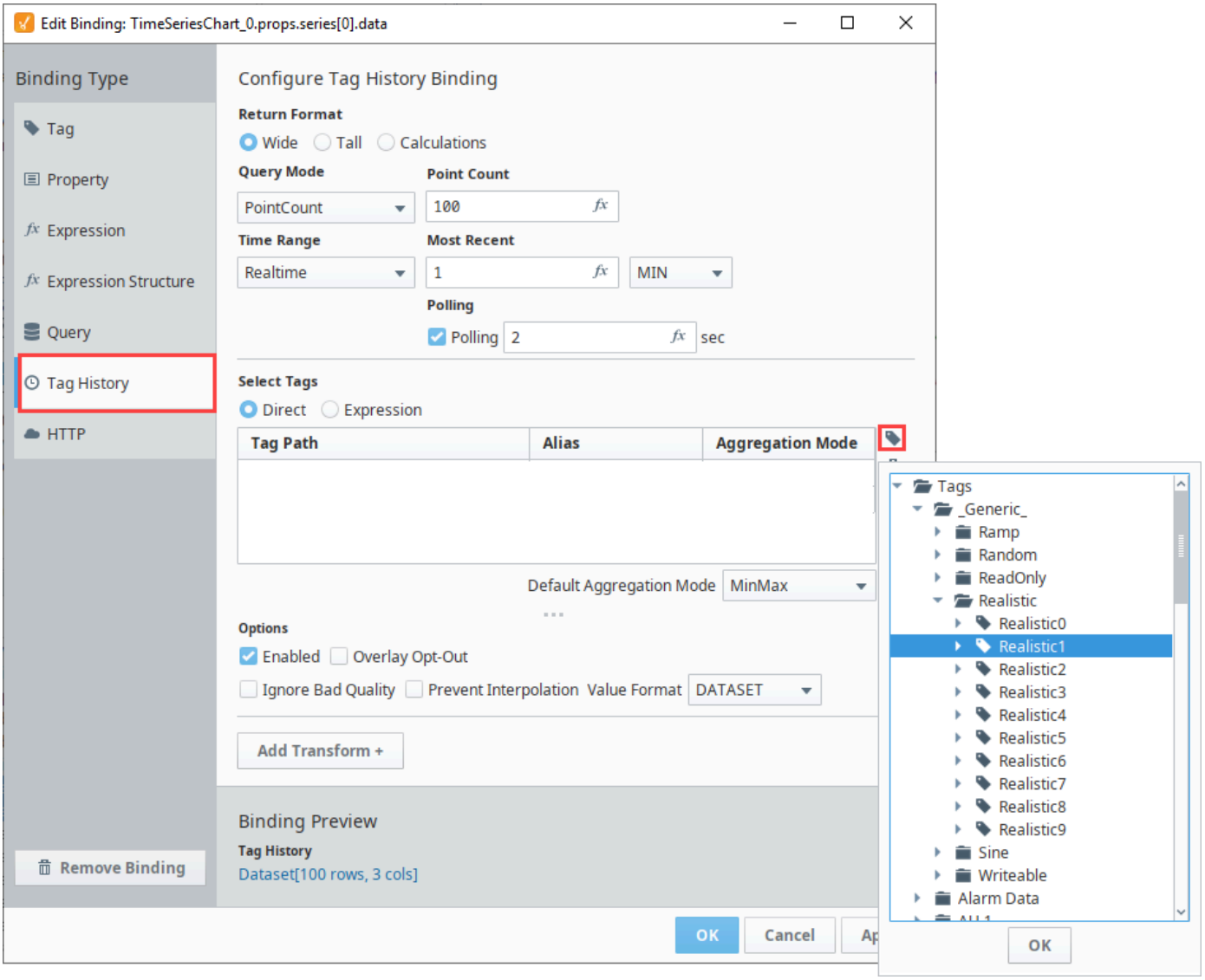


2. With the Time Series Chart selected, click on the chain link **Binding icon** under the **series > 0 > data** property.



3. This will open the Edit Binding window. Select the **Tag History** binding type.

4. From the **Select Tags** section, click on the Tag Browse icon on the far right. Browse to select your Tag path. We selected **Realistic1** and **Realistic2** from the Generic > Realistic folder.



5. Now let's configure the following Tag History binding settings:

    i. Double click in the Alias column for each Tag to add an Alias

        a. Add the **Alias** for **Realistic1** as **temp**.

b. Add the **Alias** for **Realistic2** as **pressure**.

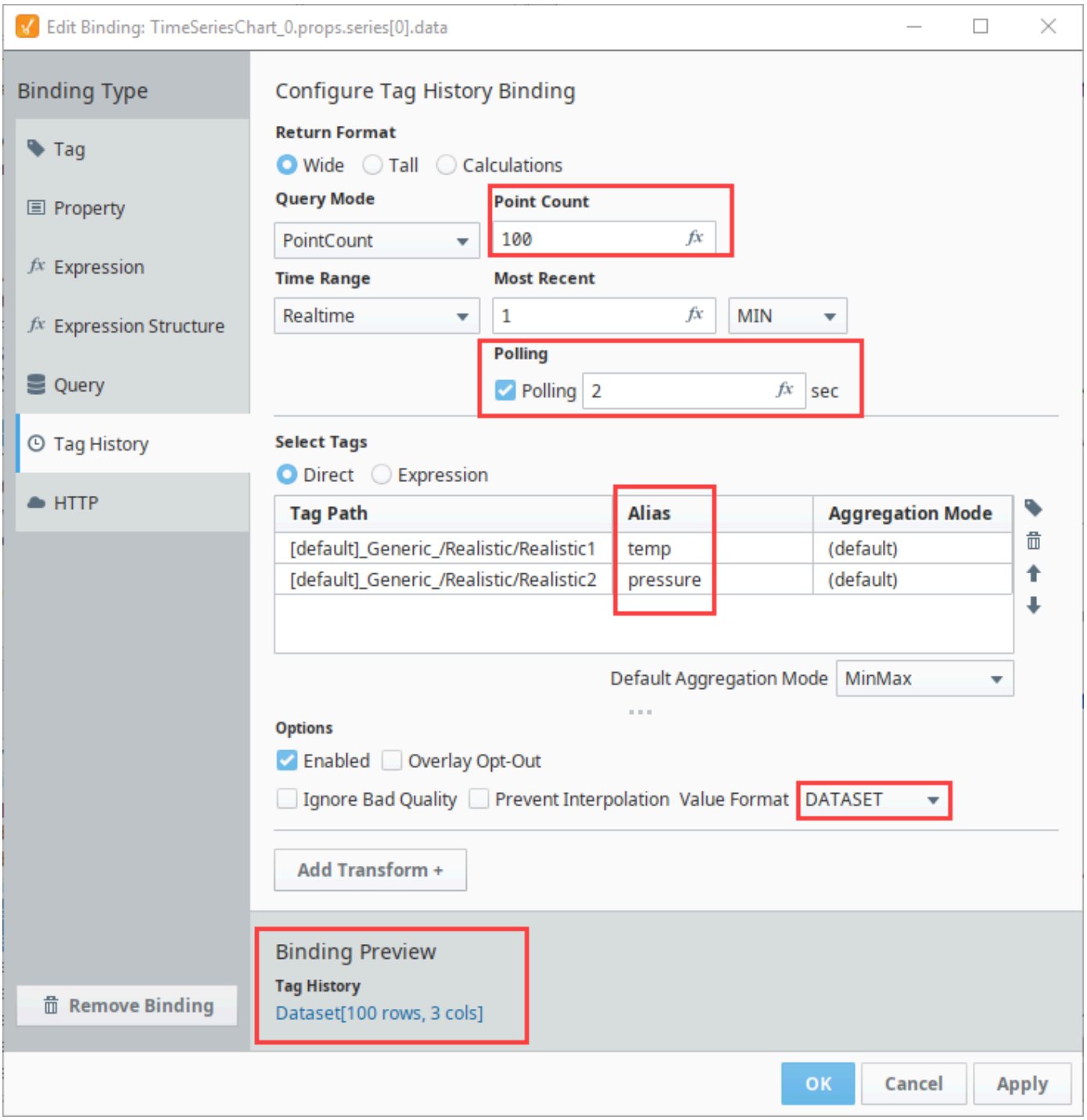
ii. Set the **Point** count to **100**.

iii. Under **Most Recent**, select **MIN** from the dropdown.

iv. Set **Polling** to **2**.

v. The Tag History binding returns a **Dataset**. Y ou will see the format type in the Binding Preview in the lower left.

6. Click **OK**.



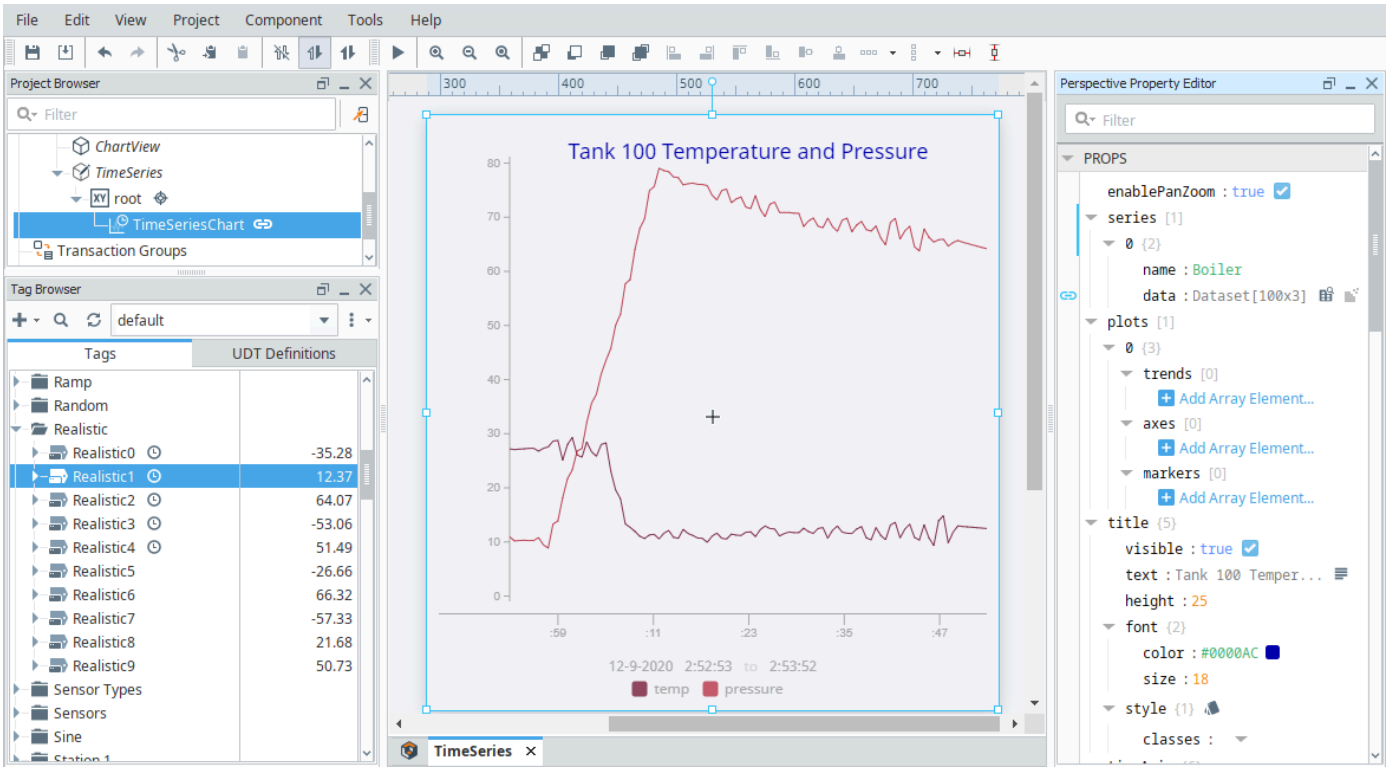
7. Now you have Tag History data in your Time Series Chart.

8. To add a title, scroll to the **Title** object in the PROPS section and set the following values:

i. **title.visible** - Set to **true**.

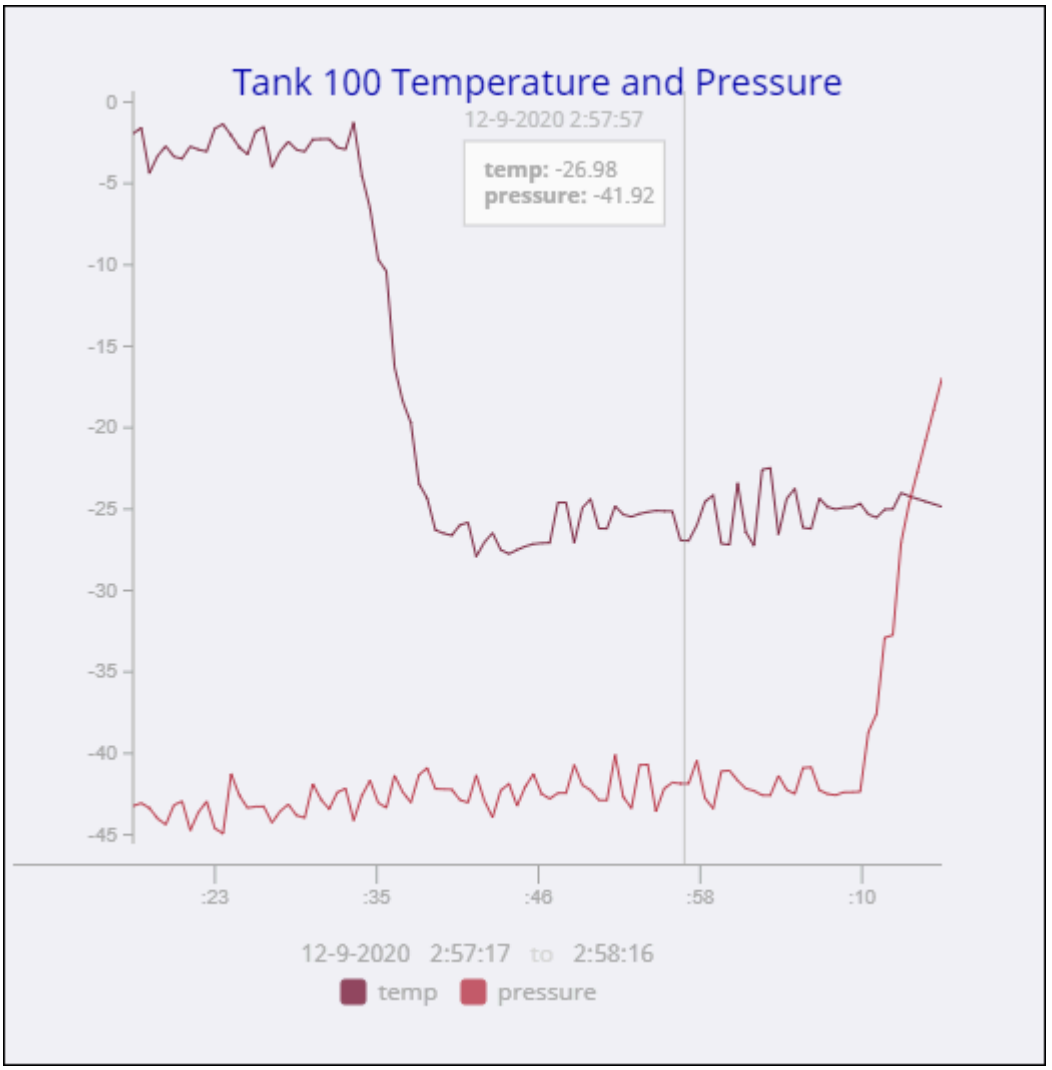
ii. **title.text** - Enter a title: **Tank 100 Pressure and Temp Tracking**.

iii. **legend.visible** - Set to true.

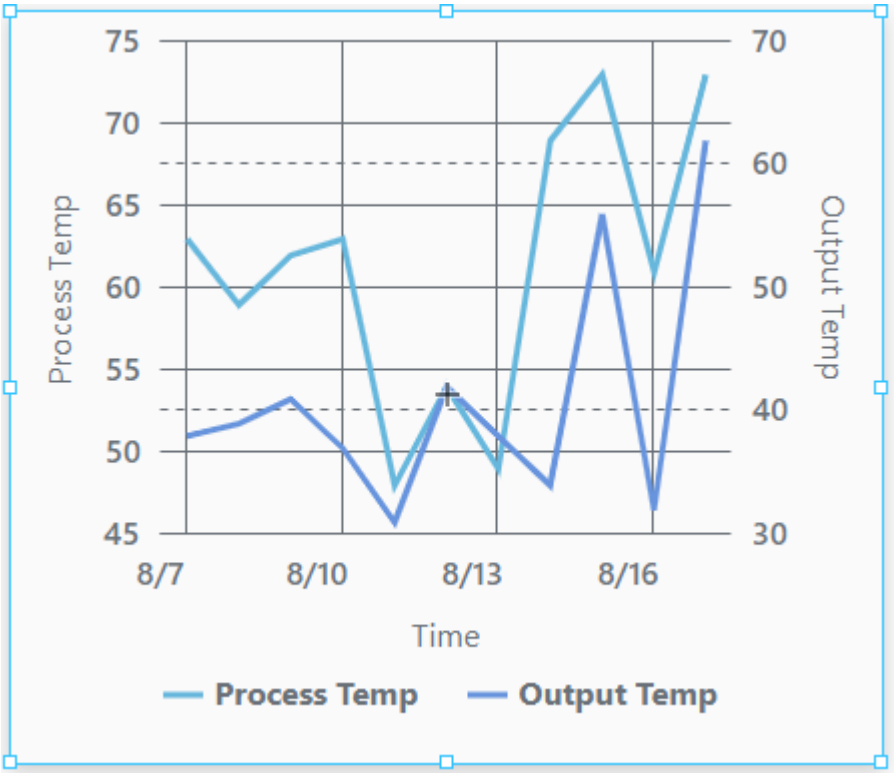


9. Save your project.

10. In **Preview Mode**, when you hover over the chart you will see a timestamp and Temp and Pressure values representing the current x-trace position.


Tank 100 Temperature and Pressure
12-9-2020 2:57:57
temp: -26.98
pressure: -41.92
0
-5
-10
-15
-20
-25
-30
-35
-40
-45
:23
:35
:46
:58
:10
12-9-2020 2:57:17 to 2:58:16
temp
pressure

Perspective - XY Chart



Component Palette Icon

XY Chart

The XY Chart displays data trends. It provides a flexible way to display either timeseries or X-Y data by entering data in the **dataSources** property. It is fully customizable in its appearance, from labels, colors, line widths, legend, scroll bars, and text styles.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
dataSources	The data source objects used to populate the chart. Each data source contains an array of records used by axes and series configurations. When using a date or time x-axis, the data must be sorted in ascending timestamp order.	Object
title	The configuration for the chart title. • **text**: The text displayed as the chart title. • **appearance**: The configuration settings that control the visual styling of the title. ◦ **color**: The color of the title text. ◦ **font**: The font settings applied to the title text. ▪ **size**: The font size of the title text in pixels or points. ▪ **weight**: The font weight of the title text. ◦ **padding**: The spacing around the title. ▪ **top**: The padding above the title. ▪ **bottom**: The padding below the title. ▪ **left**: The padding to the left of the title. ▪ **right**: The padding to the right of the title.	Object

Name	Description	Property Type
subtitle	The configuration for the chart subtitle. • **text**: The text displayed as the chart subtitle. • **appearance**: The configuration settings that control the visual styling of the subtitle. ◦ **color**: The color of the subtitle text. ◦ **font**: The font settings applied to the subtitle text. ▪ **size**: The font size of the subtitle text in pixels or points. ▪ **weight**: The font weight of the subtitle text. ◦ **padding**: The spacing around the subtitle. ▪ **top**: The padding above the subtitle. ▪ **bottom**: The padding below the subtitle. ▪ **left**: The padding to the left of the subtitle. ▪ **right**: The padding to the right of the subtitle.	Object
legend	The configuration for the chart legend. • **enabled**: Enables the display of the legend. • **position**: The position of the legend relative to the chart. • **absolute**: The position of the legend when using absolute placement. ◦ **x**: The horizontal position of the legend. ◦ **y**: The vertical position of the legend. • **markers**: The configuration settings for legend markers. ◦ **enabled**: Enables markers in the legend. ◦ **width**: The width of each marker in pixels. ◦ **height**: The height of each marker in pixels. ◦ **mirrorLookOfSeries**: Determines whether markers match the appearance of the associated series. ◦ **stroke**: The stroke settings for marker outlines. ▪ **color**: The stroke color of the marker. ▪ **width**: The stroke width in pixels. ▪ **opacity**: The stroke opacity from 0 to 1. ◦ **cornerRadius**: The corner radius applied to markers.	Object

Name	Description	Property Type
	▪ **topLeft**: The top-left corner radius. ▪ **topRight**: The top-right corner radius. ▪ **bottomLeft**: The bottom-left corner radius. ▪ **bottomRight**: The bottom-right corner radius. ◦ **icon**: The icon configuration for legend markers. ▪ **path**: The icon path in the format library/iconName. ▪ **color**: The color of the icon. ▪ **verticalCenter**: The vertical alignment of the icon. ▪ **horizontalCenter**: The horizontal alignment of the icon. ▪ **width**: The width of the icon in pixels. ▪ **height**: The height of the icon in pixels. • **labels**: The configuration settings for legend labels. ◦ **text**: The template string applied to legend labels. ◦ **font**: The font settings applied to legend labels. ▪ **weight**: The font weight of the label text. ▪ **size**: The font size of the label text. ▪ **color**: The color of the label text.	
cursor	The configuration for the chart cursor. • **enabled**: Enables cursor interactions on the chart. • **series**: The series that the cursor is bound to. • **lineX**: The configuration settings for the X axis cursor line. ◦ **enabled**: Enables the X axis cursor line. ◦ **stroke**: The stroke settings for the X axis cursor line. ▪ **color**: The color of the cursor line. ▪ **width**: The width of the cursor line in pixels. ▪ **opacity**: The opacity of the cursor line from 0 to 1. ▪ **dashArray**: The dash pattern used to render the cursor line.	Object

Name	Description	Property Type
	• **lineY**: The configuration settings for the Y axis cursor line. ◦ **enabled**: Enables the Y axis cursor line. ◦ **stroke**: The stroke settings for the Y axis cursor line. ▪ **color**: The color of the cursor line. ▪ **width**: The width of the cursor line in pixels. ▪ **opacity**: The opacity of the cursor line from 0 to 1. ▪ **dashArray**: The dash pattern used to render the cursor line. • **enableTransitions**: Enables transition animations for cursor interactions.	
enableTransitions	Enables transition animations. The default value is false.	Boolean
scrollBars	The configuration for the chart scrollbars. • **horizontal**: The configuration settings for the horizontal scrollbar. ◦ **enabled**: Enables the horizontal scrollbar. ◦ **series**: The series associated with the horizontal scrollbar. • **vertical**: The configuration settings for the vertical scrollbar. ◦ **enabled**: Enables the vertical scrollbar. ◦ **series**: The series associated with the vertical scrollbar.	Object

Name	Description	Property Type
selection	The configuration for selecting data points. Data points can be selected when `series[x].render` is set to column, candlestick, or line with bullets enabled. • **enabled**: Enables selection of bullets, columns, and candlesticks. • **data**: The array of currently selected data points. • **selectedHighlightColor**: The highlight color applied to selected data points.	Object
background	The configuration for the chart background. • **render**: The background render mode. • **gradient**: The configuration settings for gradient backgrounds. ◦ **stops**: The gradient stop definitions. ▪ **color**: The color of each gradient stop. ▪ **opacity**: The opacity of each gradient stop from 0 to 1. ◦ **rotation**: The rotation angle of the gradient. • **color**: The background color when using a solid fill. • **opacity**: The background opacity from 0 to 1.	Object
xAxes	The configuration for X axes. See the **X Axes Properties** section below for more details.	Array
yAxes	The configuration for Y axes. See the **Y Axes Properties** section below for more details.	Array
series	The array of series configurations applied to the chart. See the **Series Properties** section below for more details.	Array
style	The style configuration for this component. A full menu of **style options** is available. You can also specify a **style class**.	Object

Axes Properties

Axes define how data is mapped and displayed on the chart. Each axis configuration controls scaling, labeling, formatting, and visual presentation. Multiple axes can be defined and referenced by series to control how data is rendered.

`xAxes[x]` Properties

Name	Description	Property Type
name	A unique name used to identify this axis configuration. This value is required for series configuration.	String
label	The configuration for the axis label. • **enabled**: Enables the axis label. • **text**: The text displayed for the axis label. • **color**: The color of the axis label text.	Object
inversed	Determines whether the axis scale is inverted.	Boolean
visible	Controls the visibility of the axis.	Boolean
tooltip	The configuration for the axis tooltip. • **enabled**: Enables the tooltip. • **text**: The tooltip text as a format string. • **cornerRadius**: The corner radius of the tooltip. • **pointerLength**: The length of the tooltip pointer in pixels. • **background**: The background settings for the tooltip. ◦ **color**: The background color of the tooltip. ◦ **opacity**: The background opacity from 0 to 1.	Object
render	The axis type to render. Options include category, date, or value. The default value is date.	String
category	The configuration for category axis breaks. • **enabled**: Enables a category break range. • **startCategory**: The starting category of the break. • **endCategory**: The ending category of the break. • **size**: The size of the break as a decimal percentage of removed values.	Object

Name	Description	Property Type
date	The configuration for date-based axes. • **baseInterval**: The base interval settings for the axis. ◦ **enabled**: Enables base interval configuration. ◦ **timeUnit**: The base time unit. ◦ **count**: The number of time units per interval. ◦ **skipEmptyPeriods**: Removes empty time units from display. • **range**: The range settings for the axis. ◦ **max**: The maximum date in the range as a Unix timestamp in milliseconds. ◦ **min**: The minimum date in the range as a Unix timestamp in milliseconds. ◦ **useStrict**: Strictly enforces the configured range. • **break**: The formatting settings for date values. ◦ **inputFormat**: The date format of values provided by the data source. ◦ **format**: The date format used to display axis labels.	Object
value	The configuration for value-based axes. • **range**: The range settings for the axis. ◦ **max**: The maximum numeric value in the range. ◦ **min**: The minimum numeric value in the range. ◦ **useStrict**: Strictly enforces the configured range. • **logarithmic**: Uses logarithmic scaling for the axis. • **break**: The configuration for value breaks. ◦ **enabled**: Enables a value break range. ◦ **startValue**: The starting value of the break. ◦ **endValue**: The ending value of the break. ◦ **size**: The size of the break as a percentage of removed values. • **format**: The number format applied to axis values.	Object

Name	Description	Property Type
appearance	The configuration for the visual appearance of the axis. • **opposite**: Renders the axis on the opposite side of the chart. • **inside**: Renders axis labels on the inside of the axis. • **labels**: The label styling configuration. ◦ **color**: The color of the labels. ◦ **opacity**: The opacity of the labels from 0 to 1. ◦ **rotation**: The rotation of the labels in degrees. ◦ **wrap**: Wraps the label text. ◦ **verticalCenter**: The vertical alignment of the label. ◦ **horizontalCenter**: The horizontal alignment of the label. • **grid**: The grid line styling configuration. ◦ **color**: The color of the grid lines. ◦ **opacity**: The opacity of the grid lines from 0 to 1. ◦ **dashArray**: The dash pattern used to render the grid lines. ◦ **minDistance**: The minimum distance between grid lines in pixels. ◦ **position**: The relative position of the grid lines. • **font**: The font settings applied to axis labels. ◦ **size**: The font size of the labels. ◦ **weight**: The font weight of the labels.	Object

`yAxes[x]` Properties

Name	Description	Property Type
name	A unique name used to identify this axis configuration. This value is required for series configuration.	String
label	The configuration for the axis label. • **enabled**: Enables the axis label. • **text**: The text displayed for the axis label. • **color**: The color of the axis label text.	Object
inversed	Determines whether the axis scale is inverted.	Boolean
visible	Controls the visibility of the axis.	Boolean
tooltip	The configuration for the axis tooltip. • **enabled**: Enables the tooltip. • **text**: The tooltip text as a format string. • **cornerRadius**: The corner radius of the tooltip. • **pointerLength**: The length of the tooltip pointer in pixels. • **background**: The background settings for the tooltip. ◦ **color**: The background color of the tooltip. ◦ **opacity**: The background opacity from 0 to 1.	Object
render	The axis type to render. Options include category, date, or value. The default value is date.	String
category	The configuration for category axis breaks. • **enabled**: Enables a category break range. • **startCategory**: The starting category of the break. • **endCategory**: The ending category of the break. • **size**: The size of the break as a decimal percentage of removed values.	Object

Name	Description	Property Type
date	The configuration for date-based axes. • **baseInterval**: The base interval settings for the axis. ◦ **enabled**: Enables base interval configuration. ◦ **timeUnit**: The base time unit. ◦ **count**: The number of time units per interval. ◦ **skipEmptyPeriods**: Removes empty time units from display. • **range**: The range settings for the axis. ◦ **max**: The maximum date in the range as a Unix timestamp in milliseconds. ◦ **min**: The minimum date in the range as a Unix timestamp in milliseconds. ◦ **useStrict**: Strictly enforces the configured range. • **break**: The formatting settings for date values. ◦ **inputFormat**: The date format of values provided by the data source. ◦ **format**: The date format used to display axis labels.	Object
value	The configuration for value-based axes. • **range**: The range settings for the axis. ◦ **max**: The maximum numeric value in the range. ◦ **min**: The minimum numeric value in the range. ◦ **useStrict**: Strictly enforces the configured range. • **logarithmic**: Uses logarithmic scaling for the axis. • **break**: The configuration for value breaks. ◦ **enabled**: Enables a value break range. ◦ **startValue**: The starting value of the break. ◦ **endValue**: The ending value of the break. ◦ **size**: The size of the break as a percentage of removed values. • **format**: The number format applied to axis values.	Object

Name	Description	Property Type
appearance	The configuration for the visual appearance of the axis. • **opposite**: Renders the axis on the opposite side of the chart. • **inside**: Renders axis labels on the inside of the axis. • **labels**: The label styling configuration. ◦ **color**: The color of the labels. ◦ **opacity**: The opacity of the labels. ◦ **rotation**: The rotation of the labels. ◦ **wrap**: Wraps the label text. ◦ **verticalCenter**: The vertical alignment of the label. ◦ **horizontalCenter**: The horizontal alignment of the label. • **grid**: The grid line styling configuration. ◦ **color**: The color of the grid lines. ◦ **opacity**: The opacity of the grid lines. ◦ **dashArray**: The dash pattern used to render the grid lines. ◦ **minDistance**: The minimum distance between grid lines. ◦ **position**: The relative position of the grid lines. • **font**: The font settings applied to axis labels. ◦ **size**: The font size of the labels. ◦ **weight**: The font weight of the labels.	Object

`series[x]` Properties

Each series defines how data from a data source is rendered on the chart, including axis mapping, appearance, and interaction behavior.

Name	Description	Property Type
name	A unique name used to identify this series.	String
label	The configuration for the series label. • **text**: The text displayed for this series in the legend.	Object
visible	Enables series visibility. The default value is true.	Boolean
hiddenInLegend	Determines whether the series is hidden from the legend. The default value is false.	Boolean
defaultState	The default visibility state of the series. • **visible**: Determines whether the series is visible by default.	Object
data	The data configuration for the series. • **source**: The name of the data source used by this series. • **x**: The field or index used for X values. • **y**: The field or index used for Y values.	Object
xAxis	The name of the X axis used by this series.	String
yAxis	The name of the Y axis used by this series.	String
zIndex	The stacking order of the series relative to other series.	Numeric
tooltip	The configuration for the series tooltip. • **enabled**: Enables the tooltip. • **text**: The tooltip text as a format string. • **cornerRadius**: The corner radius of the tooltip. • **pointerLength**: The length of the tooltip pointer in pixels. • **background**: The background settings for the tooltip. ◦ **color**: The background color of the tooltip. ◦ **opacity**: The background opacity from 0 to 1.	Object

Name	Description	Property Type
render	The rendering mode for the series. Options include candlestick, column, line, or stepLine, each of which exposes its own configuration object below. The default value is line.	String

Name	Description	Property Type
candlestick	Available when **render** is set to candlestick. The configuration for candlestick-based series rendering. • **open**: The configuration for open values. ◦ **x**: The field used for open X values. ◦ **y**: The field used for open Y values. • **high**: The configuration for high values. ◦ **x**: The field used for high X values. ◦ **y**: The field used for high Y values. • **low**: The configuration for low values. ◦ **x**: The field used for low X values. ◦ **y**: The field used for low Y values. • **appearance**: The visual configuration for candlesticks. ◦ **fill**: The fill settings. ▪ **color**: The fill color of the candlestick. ▪ **opacity**: The fill opacity from 0 to 1. ◦ **stroke**: The stroke settings. ▪ **color**: The stroke color. ▪ **opacity**: The stroke opacity from 0 to 1. ▪ **width**: The stroke width in pixels. ◦ **stacked**: Determines whether the candlestick series is stacked. ◦ **deriveFieldsFromData**: Derives appearance values from the data source. ▪ **fill**: The derived fill settings. ▪ **stroke**: The derived stroke settings. ◦ **heatRules**: The configuration for heat rules. ▪ **enabled**: Enables heat rules. ▪ **max**: The maximum color value. ▪ **min**: The minimum color value. ▪ **dataField**: The data field used for heat rule calculations.	Object
column	Available when **render** is set to column. The configuration for column-based series rendering.	Object

Name	Description	Property Type
	• **open**: The configuration for open values. ◦ **x**: The field used for open X values. ◦ **y**: The field used for open Y values. • **appearance**: The visual configuration for columns. ◦ **fill**: The fill settings. ▪ **color**: The fill color of the column. ▪ **opacity**: The fill opacity from 0 to 1. ◦ **stroke**: The stroke settings. ▪ **color**: The stroke color. ▪ **opacity**: The stroke opacity from 0 to 1. ▪ **width**: The stroke width in pixels. ◦ **width**: The width of the column. ◦ **height**: The height of the column. ◦ **stacked**: Determines whether the column series is stacked. ◦ **deriveFieldsFromData**: Derives appearance values from the data source. ▪ **fill**: The derived fill settings. ▪ **stroke**: The derived stroke settings. ◦ **heatRules**: The configuration for heat rules. ▪ **max**: The maximum color value. ▪ **min**: The minimum color value. ▪ **dataField**: The data field used for heat rule calculations.	
line	Available when **render** is set to line. The configuration for line-based series rendering. • **open**: The configuration for open values. ◦ **x**: The field used for open X values. ◦ **y**: The field used for open Y values. • **appearance**: The visual configuration for the line. ◦ **connect**: Connects the line over empty data points. The default value is true.	Object

Name	Description	Property Type
	◦ **tensionX**: The horizontal tension of the line. Range is 0 to 1, where 1 produces straight segments between points and 0 produces maximum curvature. The default value is 1. ◦ **tensionY**: The vertical tension of the line. Range is 0 to 1. The default value is 1. ◦ **minDistance**: The minimum distance in pixels between two rendered points. The default value is 0.5. ◦ **stroke**: The stroke settings. ▪ **width**: The stroke width in pixels. The default value is 3. ▪ **opacity**: The stroke opacity from 0 to 1. ▪ **color**: The stroke color. ▪ **dashArray**: The dash pattern used to render the stroke. ◦ **fill**: The fill settings for the area under the line. ▪ **color**: The fill color. ▪ **opacity**: The fill opacity from 0 to 1. ◦ **bullets**: The array of bullet configurations rendered at each data point. ▪ **enabled**: Enables bullets. ▪ **render**: The bullet type. Options are circle or label. ▪ **radius**: **New in 8.3.5** The bullet radius. Replaces the previous **width** and **height** properties (see the **deprecated XY chart documentation** for the original properties). ▪ **rotation**: The bullet rotation in degrees. ▪ **label**: The label settings for the bullet. ▪ **text**: The label text. ▪ **position**: The label position. ▪ **dx**: The label X position. ▪ **dy**: The label Y position.	

Name	Description	Property Type
	▪ **truncate**: New in 8.3.5 Truncates the text if it does not fit. The default value is true. ▪ **hideOversized**: New in 8.3.5 Hides the label if it is too long. The default value is true. ▪ **fill**: The fill settings for the bullet. ▪ **stroke**: The stroke settings for the bullet. ▪ **tooltip**: The tooltip settings for the bullet. ▪ **enabled**: Enables the tooltip. ▪ **text**: The tooltip text. Can be a format string. ▪ **cornerRadius**: The corner radius. ▪ **pointerLength**: The pointer length. ▪ **getFillFromObject**: New in 8.3.5 Applies the `bullets.fill.color` value to the tooltip background. If false, the tooltip background defers to the **tooltip.background** property. The default value is true. ▪ **background**: The background color and opacity for the tooltip. ▪ **deriveFieldsFromData**: Derives bullet appearance values from the data source. ▪ **heatRules**: The configuration for heat rules applied to bullet appearance.	
stepLine	Available when **render** is set to stepLine. The configuration for step line-based series rendering. • **open**: The configuration for open values. ◦ **x**: The field used for open X values. ◦ **y**: The field used for open Y values. • **appearance**: The visual configuration for the step line. ◦ **connect**: Connects the line over empty data points. The default value is true.	Object

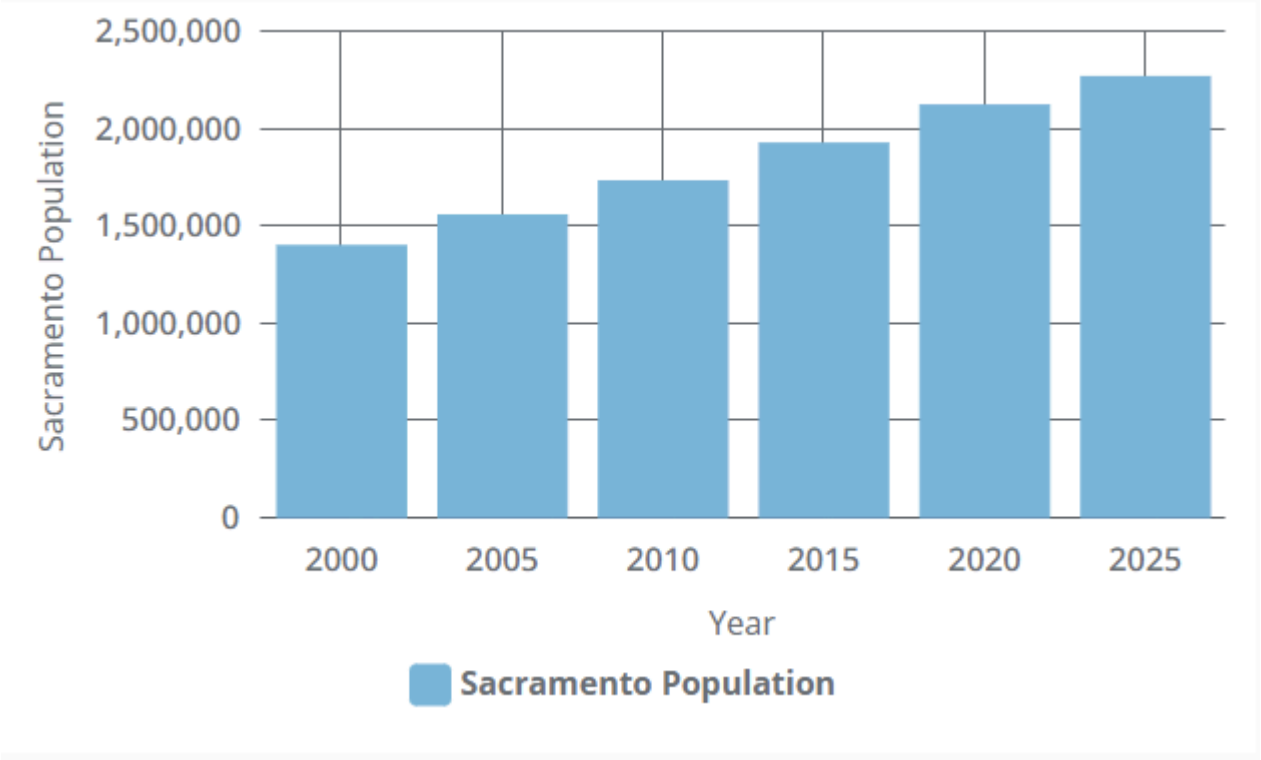
Name	Description	Property Type
	◦ **tensionX**: The horizontal tension of the line. Range is 0 to 1. The default value is 1. ◦ **tensionY**: The vertical tension of the line. Range is 0 to 1. The default value is 1. ◦ **minDistance**: The minimum distance in pixels between two rendered points. The default value is 0.5. ◦ **stroke**: The stroke settings. ▪ **width**: The stroke width in pixels. The default value is 3. ▪ **opacity**: The stroke opacity from 0 to 1. ▪ **color**: The stroke color. ▪ **dashArray**: The dash pattern used to render the stroke. ◦ **fill**: The fill settings for the area under the line. ▪ **color**: The fill color. ▪ **opacity**: The fill opacity from 0 to 1. ◦ **bullets**: The array of bullet configurations rendered at each data point. ▪ **enabled**: Enables bullets. ▪ **render**: The bullet type. Options are circle or label. ▪ **width**: The bullet width. ▪ **height**: The bullet height. ▪ **rotation**: The bullet rotation in degrees. ▪ **label**: The label settings for the bullet. ▪ **fill**: The fill settings for the bullet. ▪ **stroke**: The stroke settings for the bullet. ▪ **tooltip**: The tooltip settings for the bullet. ▪ **deriveFieldsFromData**: Derives bullet appearance values from the data source. ▪ **heatRules**: The configuration for heat rules applied to bullet appearance.	

Component Events

Perspective Component Events The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

XY Chart Example - Bar Chart

This example demonstrates how to configure a bar chart using the XY Chart component.



Configuring a Bar Chart Example

In this example, we'll configure the XY Chart as a bar chart and display population data from the city of Sacramento every five years since 2000. We'll start by changing our chart to use bars instead of lines.

1. Create an XY Chart in the Designer and select it.
2. In the **Perspective Property Editor**, navigate to the **series.0.render** property and change the type from **line** to **column**.
3. We'll now have our bar chart display our data. Copy the JSON content below, and paste it onto the chart component's **dataSources.example** property.

```
[
    {
        "year": "2000",
        "sacramento": 1401000
    },
    {
        "year": "2005",
        "sacramento": 1558000
    },
    {
        "year": "2010",
        "sacramento": 1733000
    },
    {
        "year": "2015",
        "sacramento": 1927000
    },
    {
        "year": "2020",
        "sacramento": 2123000
    },
    {
        "year": "2025",
        "sacramento": 2269000
    }
]
```


4. We will now need to adjust several properties to reflect our JSON content so that our data will render. Update the settings under the **xAxes.0** property to the corresponding values listed below:

    - **name**: year
    - **label.text**: Year
    - **date.baseInterval.enabled**: true
    - **date.baseInterval.timeUnit**: year
    - **date.baseInterval.skipEmptyPeriods**: true
    - **date.inputFormat**: yyyy
    - **date.format**: yyyy

5. Update the settings under the **yAxes.0** property to the corresponding values listed below:

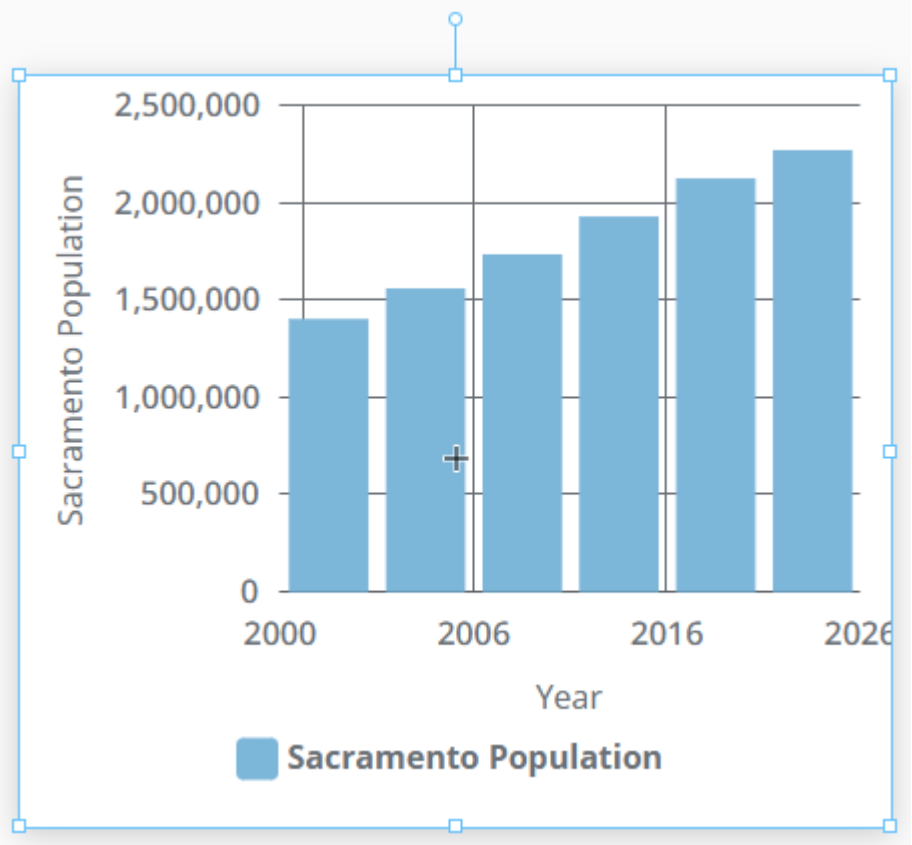
    - **name**: sacramento
    - **label.text**: Sacramento Population
    - **value.range.min**: 0

6. Update the settings under the **series.0** property to the corresponding values listed below:

    - **name**: sacramento
    - **label.text**: Sacramento Population
    - **data.x**: year
    - **data.y**: sacramento
    - **xAxis**: year
    - **yAxis**: sacramento

7. Next, navigate to the **series.1** and **yAxes.1** properties and delete them. We will not need a second series of data for this example.

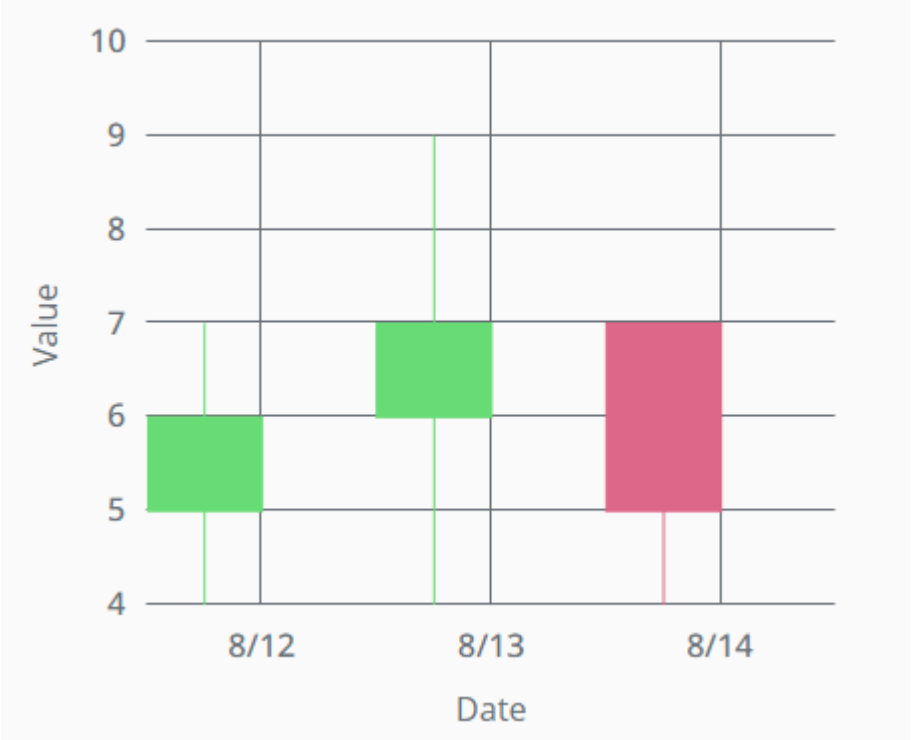
8. You may notice that the data rendered is off-center, with bars not aligning with their year values.



    To fix this, use the handles on the XY Chart to widen the component. This will resize the chart so that the data is properly displayed.

XY Chart Example - Candlestick Chart

This example demonstrates how to configure a candlestick chart. The candlestick chart needs five keys per entry on the chart: a key to represent the date of the entry and four keys that make up the candlestick. These four keys are called open, close, high, and low. These dedicated key properties are visible under the **series** when the render selection is set to candlestick, and therefore this chart does not require that the keys under **dataSources** use the exact naming conventions as shown on this page.



Configure a Candlestick Chart

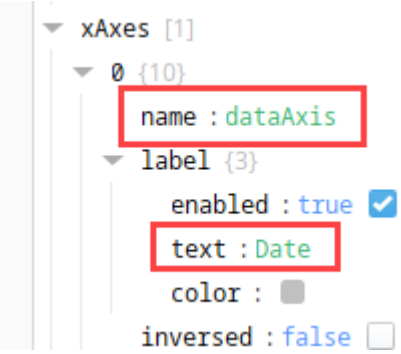
1. Create an XY chart component and select it.
2. Copy the JSON content below, and paste it onto the chart component's **dataSources.example** property.

```
[
{
    "close": 6,
    "date": "2021-8-12 15:37:19",
    "high": 7,
    "low": 2,
    "open": 5
},
{
    "close": 7,
    "date": "2021-8-13 15:37:19",
    "high": 9,
    "low": 5,
    "open": 6
},
{
    "close": 5,
    "date": "2021-8-14 15:37:19",
    "high": 7,
    "low": 4,
    "open": 7
}
]
```


3. Remove the legend by setting **legend.enabled** to false.

4. Rename **xAxes.0.name** to `dateAxis`.

5. Set **xAxes.0.label.text** to `Date`, to change the label on the x-axis.



6. Our chart only needs a single y-axis, so delete **yAxes.1**.

7. Set **yAxes.0.name** to `valuation`.

8. Set **yAxes.0.label.text** to `Value`.

9. Delete **series.1** since we only need a single series for this example.

10. Navigate to the **series.0.data** property and make sure the **source** is still `example` from the JSON string pasted in step 2.

11. Set **series.0.data.x** to the `date` key

12. Set **series.0.data.y** to the `close` key.

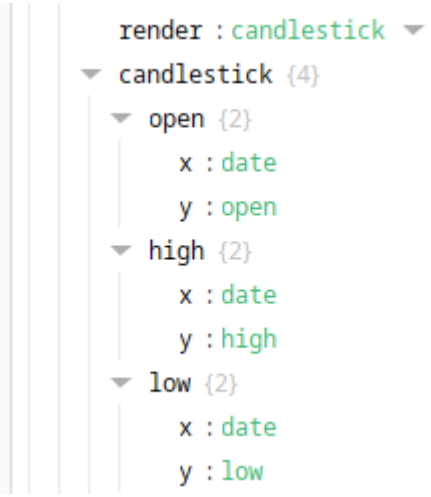
13. Set **series.0.xAxis** to the xAxes: `dateAxis`.

14. Set **series.0.yAxis** to the yAxes: `valuation`.

15. Set **series.0.render** to `candlestick`.

16. Expand the now visible candlestick properties and set the open, high, and low keys:

    - **series.0.candlestick.open**: set x to `date` and y to `open`.
    - **series.0.candlestick.high**: set x to `date` and y to `high`.
    - **series.0.candlestick.low**: set x, to `date` and y to `low`.



Your candlestick chart is now ready. Note that the third entry is in red, while the first two are green. This is because that entry's close value is less than it's open value.

Example Configuration

The JSON string below can be used to replicate the example above. Simply copy the contents of the code block and paste it into a container in your Designer.

```
[
  {
    "type": "ia.chart.xy",
    "version": 0,
    "props": {
      "legend": {
        "enabled": false
      },
      "xAxes": [
        {
          "name": "dateAxis",
          "label": {
            "enabled": true,
            "text": "Date",
            "color": ""
          },
          "visible": true,
          "tooltip": {
            "enabled": true,
            "text": "",
            "cornerRadius": 3,
            "pointerLength": 4,
            "background": {
              "color": "",
              "opacity": 1
            }
          },
          "inversed": false,
          "render": "date",
          "category": {
            "break": {
              "enabled": false,
              "startCategory": "",
              "endCategory": "",
              "size": 0.05
            }
          },
          "date": {
            "baseInterval": {
              "enabled": false,
              "timeUnit": "hour",
              "count": 1,
              "skipEmptyPeriods": false
            },
            "range": {
              "max": "",
              "min": "",
              "useStrict": false
```

```
        },
        "break": {
          "enabled": false,
          "startDate": "",
          "endDate": "",
          "size": 0.05
        },
        "inputFormat": "yyyy-MM-dd kk:mm:ss",
        "format": "M/d"
      },
      "value": {
        "range": {
          "max": "",
          "min": "",
          "useStrict": false
        },
        "logarithmic": false,
        "break": {
          "enabled": false,
          "startValue": 0,
          "endValue": 100,
          "size": 0.05
        },
        "format": "#,###.##"
      },
      "appearance": {
        "opposite": false,
        "inside": false,
        "labels": {
          "color": "",
          "opacity": 1,
          "rotation": 0,
          "verticalCenter": "middle",
          "horizontalCenter": "middle"
        },
        "grid": {
          "color": "",
          "opacity": 1,
          "dashArray": "",
          "minDistance": 60,
          "position": 0.5
        },
        "font": {
          "size": "",
          "weight": 500
        }
      }
    }
  ],
```

```
    "yAxes": [
      {
        "name": "valuation",
        "label": {
          "enabled": true,
          "text": "Value",
          "color": ""
        },
        "visible": true,
        "tooltip": {
          "enabled": true,
          "text": "",
          "cornerRadius": 3,
          "pointerLength": 4,
          "background": {
            "color": "",
            "opacity": 1
          }
        },
        "inversed": false,
        "render": "value",
        "category": {
          "break": {
            "enabled": false,
            "startCategory": "",
            "endCategory": "",
            "size": 0.05
          }
        },
        "date": {
          "baseInterval": {
            "enabled": false,
            "timeUnit": "hour",
            "count": 1,
            "skipEmptyPeriods": false
          },
          "range": {
            "max": "",
            "min": "",
            "useStrict": false
          },
          "break": {
            "enabled": false,
            "startDate": "",
            "endDate": "",
            "size": 0.05
          },
          "inputFormat": "yyyy-MM-dd kk:mm:ss",
          "format": "M/d/yyyy HH:mm:ss"
```

```
      },
      "value": {
        "range": {
          "max": "",
          "min": "",
          "useStrict": false
        },
        "logarithmic": false,
        "break": {
          "enabled": false,
          "startValue": 0,
          "endValue": 100,
          "size": 0.05
        },
        "format": "#,###.##"
      },
      "appearance": {
        "opposite": false,
        "inside": false,
        "labels": {
          "color": "",
          "opacity": 1,
          "rotation": 0,
          "verticalCenter": "middle",
          "horizontalCenter": "middle"
        },
        "grid": {
          "color": "",
          "opacity": 1,
          "dashArray": "",
          "minDistance": null,
          "position": 0.5
        },
        "font": {
          "size": "",
          "weight": 500
        }
      }
    }
  ],
  "series": [
    {
      "name": "process temp",
      "label": {
        "text": "Process Temp"
      },
      "visible": true,
      "hiddenInLegend": false,
      "defaultState": {
```

```
      "visible": true
    },
    "data": {
      "source": "example",
      "x": "date",
      "y": "close"
    },
    "xAxis": "dateAxis",
    "yAxis": "valuation",
    "zIndex": 0,
    "tooltip": {
      "enabled": true,
      "text": "{name}: [bold]{valueY}[/]",
      "cornerRadius": 3,
      "pointerLength": 4,
      "background": {
        "color": "",
        "opacity": 1
      }
    },
    "render": "candlestick",
    "candlestick": {
      "open": {
        "x": "date",
        "y": "open"
      },
      "high": {
        "x": "date",
        "y": "high"
      },
      "low": {
        "x": "date",
        "y": "low"
      },
      "appearance": {
        "fill": {
          "color": "",
          "opacity": 1
        },
        "stroke": {
          "color": "",
          "opacity": 1,
          "width": 1
        },
        "stacked": false,
        "deriveFieldsFromData": {
          "fill": {
            "color": "",
            "opacity": ""
```

```
        },
        "stroke": {
          "color": "",
          "opacity": "",
          "width": ""
        }
      },
      "heatRules": {
        "enabled": false,
        "max": "",
        "min": "",
        "dataField": ""
      }
    }
  },
  "column": {
    "open": {
      "x": "",
      "y": ""
    },
    "appearance": {
      "fill": {
        "color": "",
        "opacity": 1
      },
      "stroke": {
        "color": "",
        "opacity": 1,
        "width": 1
      },
      "stacked": false,
      "width": null,
      "height": null,
      "deriveFieldsFromData": {
        "fill": {
          "color": "",
          "opacity": ""
        },
        "stroke": {
          "color": "",
          "opacity": "",
          "width": ""
        }
      },
      "heatRules": {
        "enabled": false,
        "max": "",
        "min": "",
        "dataField": ""
```

```
      }
    }
  },
  "line": {
    "open": {
      "x": "",
      "y": ""
    },
    "appearance": {
      "connect": true,
      "tensionX": 1,
      "tensionY": 1,
      "minDistance": 0.5,
      "stroke": {
        "width": 3,
        "opacity": 1,
        "color": "",
        "dashArray": ""
      },
      "fill": {
        "opacity": 0,
        "color": ""
      },
      "bullets": [
        {
          "enabled": false,
          "render": "circle",
          "label": {
            "text": "{value}",
            "position": {
              "dx": 0,
              "dy": 0
            },
            "truncate": true,
            "hideOversized": true
          },
          "fill": {
            "color": "",
            "opacity": 1
          },
          "stroke": {
            "color": "",
            "opacity": 1,
            "width": 1
          },
          "rotation": 0,
          "radius": 5,
          "tooltip": {
            "enabled": true,
```

```
            "text": "{name}: [bold]{valueY}[/]",
            "cornerRadius": 3,
            "pointerLength": 4,
            "getFillFromObject": true,
            "background": {
              "color": "",
              "opacity": 1
            }
          },
          "deriveFieldsFromData": {
            "fill": {
              "color": "",
              "opacity": ""
            },
            "stroke": {
              "color": "",
              "opacity": "",
              "width": ""
            },
            "rotation": ""
          },
          "heatRules": {
            "enabled": false,
            "max": 100,
            "min": 2,
            "dataField": ""
          },
          "width": 10,
          "height": 10
        }
      ]
    }
  },
  "stepLine": {
    "open": {
      "x": "",
      "y": ""
    },
    "appearance": {
      "connect": true,
      "tensionX": 1,
      "tensionY": 1,
      "minDistance": 0.5,
      "stroke": {
        "width": 3,
        "opacity": 1,
        "color": "",
        "dashArray": ""
      },
```

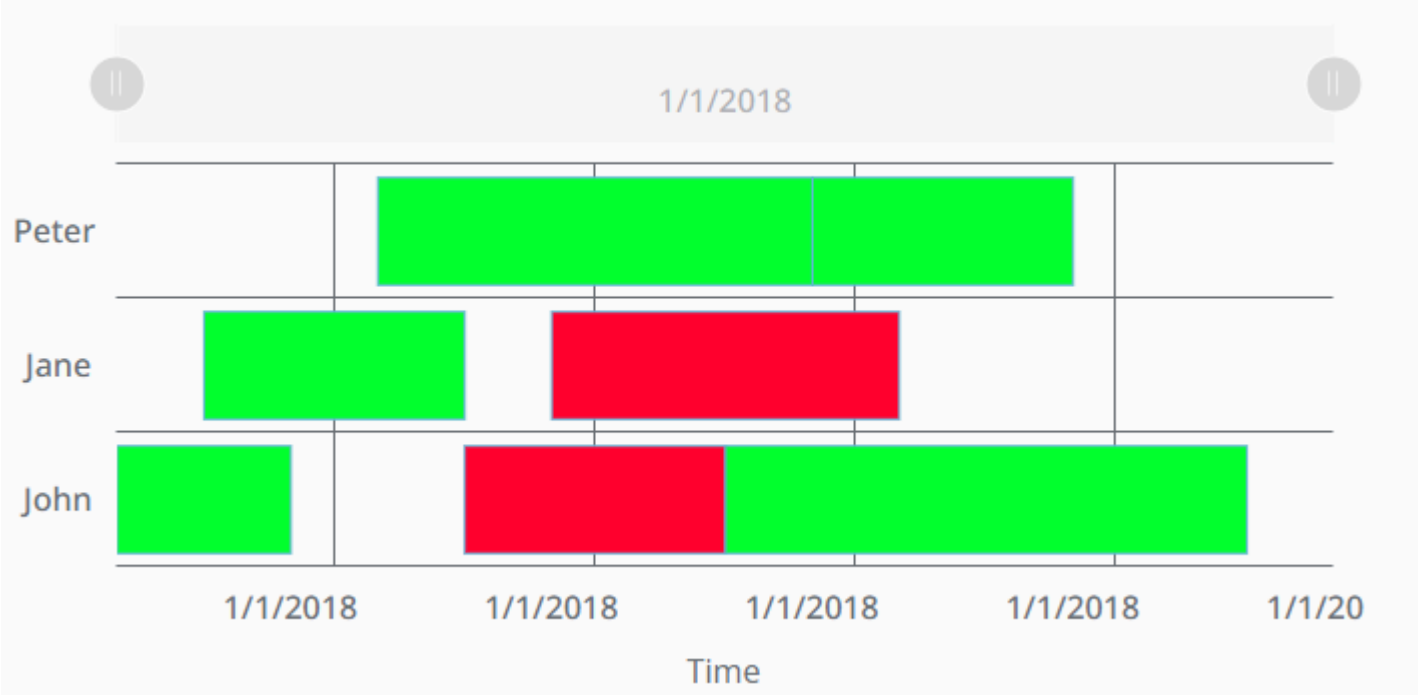
```
        "fill": {
          "opacity": 0,
          "color": ""
        },
        "bullets": [
          {
            "enabled": true,
            "render": "circle",
            "label": {
              "text": "{value}",
              "position": {
                "dx": 0,
                "dy": 0
              },
              "truncate": true,
              "hideOversized": true
            },
            "fill": {
              "color": "",
              "opacity": 1
            },
            "stroke": {
              "color": "",
              "opacity": 1,
              "width": 1
            },
            "rotation": 0,
            "radius": 5,
            "tooltip": {
              "enabled": true,
              "text": "{name}: [bold]{valueY}[/]",
              "cornerRadius": 3,
              "pointerLength": 4,
              "getFillFromObject": true,
              "background": {
                "color": "",
                "opacity": 1
              }
            },
            "deriveFieldsFromData": {
              "fill": {
                "color": "",
                "opacity": ""
              },
              "stroke": {
                "color": "",
                "opacity": "",
                "width": ""
              },
```

```
                  "rotation": ""
                },
                "heatRules": {
                  "enabled": false,
                  "max": 100,
                  "min": 2,
                  "dataField": ""
                },
                "width": 10,
                "height": 10
              }
            ]
          }
        }
      ],
      "dataSources": {
        "example": [
          {
            "close": 6,
            "date": "2021-8-12 15:37:19",
            "high": 7,
            "low": 2,
            "open": 5
          },
          {
            "close": 7,
            "date": "2021-8-13 15:37:19",
            "high": 9,
            "low": 5,
            "open": 6
          },
          {
            "close": 5,
            "date": "2021-8-14 15:37:19",
            "high": 7,
            "low": 4,
            "open": 7
          }
        ]
      }
    },
    "meta": {
      "name": "XYChart_0"
    },
    "position": {
      "x": 109,
      "y": 17,
      "height": 375,
```

```
      "width": 435
    },
    "custom": {}
  }
]
```

XY Chart Example - Gantt Chart

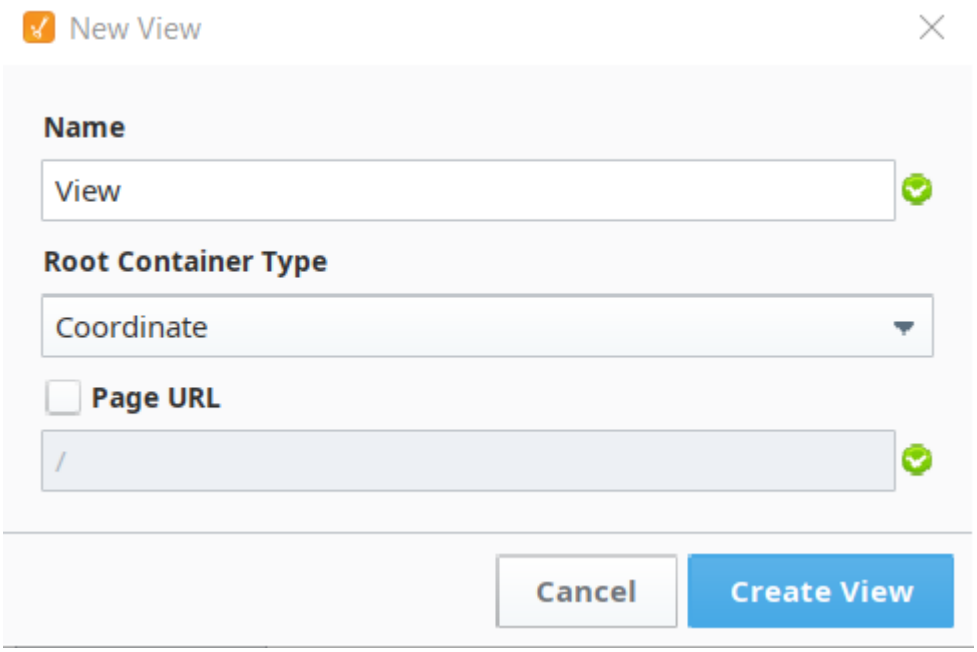
In addition to basic data plotting, an XY Chart can be used to plot horizontal bar across different lanes, in this fashion:



This can useful when it comes to machine downtime scheduling, shift scheduling, and maintenance scheduling. To achieve this functionality with the XY Chart, follow the example below:

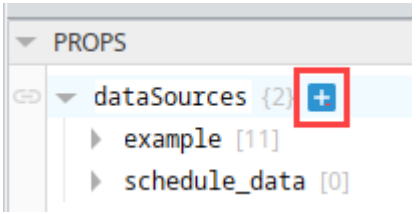
1. From the Perspective section of the Project Browser on your Designer, right click on the Views folder and select **New View...** to create a new view.

2. This will bring up the New View window. Give your view a name and select the Coordinate Root Container Type. The Page URL setting will remain unchecked for this example.



3. From the Perspective Component Palette, drag and drop a XY Chart onto your newly created view.

4. Click to select your newly added XY Chart and from the property editor, click on the + icon next to the **dataSources** property and select Array to create a new data source array and name it schedule_data.



5. Right-click on your newly created schedule_data dataSource and paste the following:

```
[
{
    "name": "John",
    "fromDate": "2018-01-01 08:00",
    "toDate": "2018-01-01 10:00",
    "color": "#00FF30"
},
{
    "name": "John",
    "fromDate": "2018-01-01 12:00",
    "toDate": "2018-01-01 15:00",
    "color": "#FF0030"
},
{
    "name": "John",
    "fromDate": "2018-01-01 15:30",
    "toDate": "2018-01-01 21:30",
    "color": "#00FF30"
},
{
    "name": "Jane",
    "fromDate": "2018-01-01 09:00",
    "toDate": "2018-01-01 12:00",
    "color": "#00FF30"
},
{
    "name": "Jane",
    "fromDate": "2018-01-01 13:00",
    "toDate": "2018-01-01 17:00",
    "color": "#FF0030"
},
{
    "name": "Peter",
    "fromDate": "2018-01-01 11:00",
    "toDate": "2018-01-01 16:00",
    "color": "#00FF30"
},
{
    "name": "Peter",
    "fromDate": "2018-01-01 16:00",
    "toDate": "2018-01-01 19:00",
    "color": "#00FF30"
}
]
```


NOTE

> Any data that you wish to plot on the XY Chart to build a Gantt Chart needs to come in the format specified by the **schedule_data** property above.

6. Set XY Chart's variables **cursor.series** and **scrollbars.horizontal.series** to "schedule data".

7. Set **legend.enabled** to false, to disable the legend.

8. Set the X axes properties as follows:

    i. Configure the X axes date format by setting the property **xAxes[0].date.format** to be a date format that works for you. For this example, we chose **date** from the dropdown, which sets it to "M/d/yyyy".
    ii. Set the properties **xAxes[0].appearance.grid.minDistance** to NULL.

9. Set the Y axes properties as follows:

    i. Delete the **yAxes[1]** property since it will not be used.
    ii. Set the property **yAxes[0].name** to "Operator".
    iii. Disable the Y axes label by setting the **yAxes[0].label.enabled** property to false.
    iv. Set the property **yAxes[0].render** to "category".
    v. Set your **yAxes[0].appearance.grid.position** to "0".

10. Set the series properties as follows:

    i. Delete the **series[1]** property since it will not be used.
    ii. Set your **series[0].name** to "schedule data". This links your Horizontal Scroll bar from step 6 to your chart's series.
    iii. Set your **series[0].data.source** to "schedule_data". This links your chart's series to your data source from step 4.
    iv. Configure your **series[0].data.x** to be "toDate" and your **series[0].data.y** to be "name".
    v. Configure your **series[0].xAxis** to be "time" and your **series[0].yAxis** to be "Operator".
    vi. Set your **series[0].render** property to "column".
    vii. Your **series[0].column.open.x** property must be set to "fromDate".
    viii. To help with data visualization, set your **series[0].tooltip.text** to `{name}: [bold] {fromDate} - {toDate}[/]`.

11. Our data includes a key for color, but our chart isn't currently using it. Set **series[0].column.appearance.deriveFieldsFromData.fill.color** to "color". This will map the "color" key in our data source to each block in the chart.

12. Save your project.

13. Put the Designer into Preview mode to see the chart in action.

Example Configuration

The JSON string below can be used to replicate the above example. Copy the contents of the code block below and paste it into a container in your Designer to populate the chart.

```
[
  {
    "type": "ia.chart.xy",
    "version": 0,
    "props": {
      "legend": {
        "enabled": false
      },
      "cursor": {
        "series": "mySeries"
      },
      "scrollBars": {
        "horizontal": {
          "series": "mySeries"
        }
      },
      "xAxes": [
        {
          "name": "time",
          "label": {
            "enabled": true,
            "text": "Time",
            "color": ""
          },
          "visible": true,
          "tooltip": {
            "enabled": true,
            "text": "",
            "cornerRadius": 3,
            "pointerLength": 4,
            "background": {
              "color": "",
              "opacity": 1
            }
          },
          "inversed": false,
          "render": "date",
          "category": {
            "break": {
              "enabled": false,
              "startCategory": "",
              "endCategory": "",
              "size": 0.05
            }
          },
          "date": {
            "baseInterval": {
              "enabled": false,
```

```
      "timeUnit": "hour",
      "count": 1,
      "skipEmptyPeriods": false
    },
    "range": {
      "max": "",
      "min": "",
      "useStrict": false
    },
    "break": {
      "enabled": false,
      "startDate": "",
      "endDate": "",
      "size": 0.05
    },
    "inputFormat": "yyyy-MM-dd kk:mm:ss",
    "format": "M/d/yyyy"
  },
  "value": {
    "range": {
      "max": "",
      "min": "",
      "useStrict": false
    },
    "logarithmic": false,
    "break": {
      "enabled": false,
      "startValue": 0,
      "endValue": 100,
      "size": 0.05
    },
    "format": "#,###.##"
  },
  "appearance": {
    "opposite": false,
    "inside": false,
    "labels": {
      "color": "",
      "opacity": 1,
      "rotation": 0,
      "verticalCenter": "middle",
      "horizontalCenter": "middle"
    },
    "grid": {
      "color": "",
      "opacity": 1,
      "dashArray": "",
      "minDistance": 60,
      "position": 0.5
```

```
        },
        "font": {
          "size": "",
          "weight": 500
        }
      }
    }
  ],
  "yAxes": [
    {
      "name": "Operator",
      "label": {
        "enabled": false,
        "text": "Process Temp",
        "color": ""
      },
      "visible": true,
      "tooltip": {
        "enabled": true,
        "text": "",
        "cornerRadius": 3,
        "pointerLength": 4,
        "background": {
          "color": "",
          "opacity": 1
        }
      },
      "inversed": false,
      "render": "category",
      "category": {
        "break": {
          "enabled": false,
          "startCategory": "",
          "endCategory": "",
          "size": 0.05
        }
      },
      "date": {
        "baseInterval": {
          "enabled": false,
          "timeUnit": "hour",
          "count": 1,
          "skipEmptyPeriods": false
        },
        "range": {
          "max": "",
          "min": "",
          "useStrict": false
        },
```

```
          "break": {
            "enabled": false,
            "startDate": "",
            "endDate": "",
            "size": 0.05
          },
          "inputFormat": "yyyy-MM-dd kk:mm:ss",
          "format": "M/d/yyyy HH:mm:ss"
        },
        "value": {
          "range": {
            "max": "",
            "min": "",
            "useStrict": false
          },
          "logarithmic": false,
          "break": {
            "enabled": false,
            "startValue": 0,
            "endValue": 100,
            "size": 0.05
          },
          "format": "#,###.##"
        },
        "appearance": {
          "opposite": false,
          "inside": false,
          "labels": {
            "color": "",
            "opacity": 1,
            "rotation": 0,
            "verticalCenter": "middle",
            "horizontalCenter": "middle"
          },
          "grid": {
            "color": "",
            "opacity": 1,
            "dashArray": "",
            "minDistance": null,
            "position": 0
          },
          "font": {
            "size": "",
            "weight": 500
          }
        }
      }
    ],
    "series": [
```

```
{
  "name": "mySeries",
  "label": {
    "text": "Process Temp"
  },
  "visible": true,
  "hiddenInLegend": false,
  "defaultState": {
    "visible": true
  },
  "data": {
    "source": "example",
    "x": "toDate",
    "y": "name"
  },
  "xAxis": "time",
  "yAxis": "Operator",
  "zIndex": 0,
  "tooltip": {
    "enabled": true,
    "text": "{name}: [bold]{fromDate} - {toDate}[/]",
    "cornerRadius": 3,
    "pointerLength": 4,
    "background": {
      "color": "",
      "opacity": 1
    }
  },
  "render": "column",
  "candlestick": {
    "open": {
      "x": "",
      "y": ""
    },
    "high": {
      "x": "",
      "y": ""
    },
    "low": {
      "x": "",
      "y": ""
    },
    "appearance": {
      "fill": {
        "color": "",
        "opacity": 1
      },
      "stroke": {
        "color": "",
```

```
            "opacity": 1,
            "width": 1
          },
          "stacked": false,
          "deriveFieldsFromData": {
            "fill": {
              "color": "",
              "opacity": ""
            },
            "stroke": {
              "color": "",
              "opacity": "",
              "width": ""
            }
          },
          "heatRules": {
            "enabled": false,
            "max": "",
            "min": "",
            "dataField": ""
          }
        }
      },
      "column": {
        "open": {
          "x": "fromDate",
          "y": ""
        },
        "appearance": {
          "fill": {
            "color": "",
            "opacity": 1
          },
          "stroke": {
            "color": "",
            "opacity": 1,
            "width": 1
          },
          "stacked": false,
          "width": null,
          "height": null,
          "deriveFieldsFromData": {
            "fill": {
              "color": "color",
              "opacity": ""
            },
            "stroke": {
              "color": "",
              "opacity": "",
```

```
            "width": ""
          }
        },
        "heatRules": {
          "enabled": false,
          "max": "",
          "min": "",
          "dataField": ""
        }
      }
    },
    "line": {
      "open": {
        "x": "",
        "y": ""
      },
      "appearance": {
        "connect": true,
        "tensionX": 1,
        "tensionY": 1,
        "minDistance": 0.5,
        "stroke": {
          "width": 3,
          "opacity": 1,
          "color": "",
          "dashArray": ""
        },
        "fill": {
          "opacity": 0,
          "color": ""
        },
        "bullets": [
          {
            "enabled": false,
            "render": "circle",
            "label": {
              "text": "{value}",
              "position": {
                "dx": 0,
                "dy": 0
              },
              "truncate": true,
              "hideOversized": true
            },
            "fill": {
              "color": "",
              "opacity": 1
            },
            "stroke": {
```

```
            "color": "",
            "opacity": 1,
            "width": 1
          },
          "rotation": 0,
          "radius": 5,
          "tooltip": {
            "enabled": true,
            "text": "{name}: [bold]{valueY}[/]",
            "cornerRadius": 3,
            "pointerLength": 4,
            "getFillFromObject": true,
            "background": {
              "color": "",
              "opacity": 1
            }
          },
          "deriveFieldsFromData": {
            "fill": {
              "color": "",
              "opacity": ""
            },
            "stroke": {
              "color": "",
              "opacity": "",
              "width": ""
            },
            "rotation": ""
          },
          "heatRules": {
            "enabled": false,
            "max": 100,
            "min": 2,
            "dataField": ""
          },
          "width": 10,
          "height": 10
        }
      ]
    }
  },
  "stepLine": {
    "open": {
      "x": "",
      "y": ""
    },
    "appearance": {
      "connect": true,
      "tensionX": 1,
```

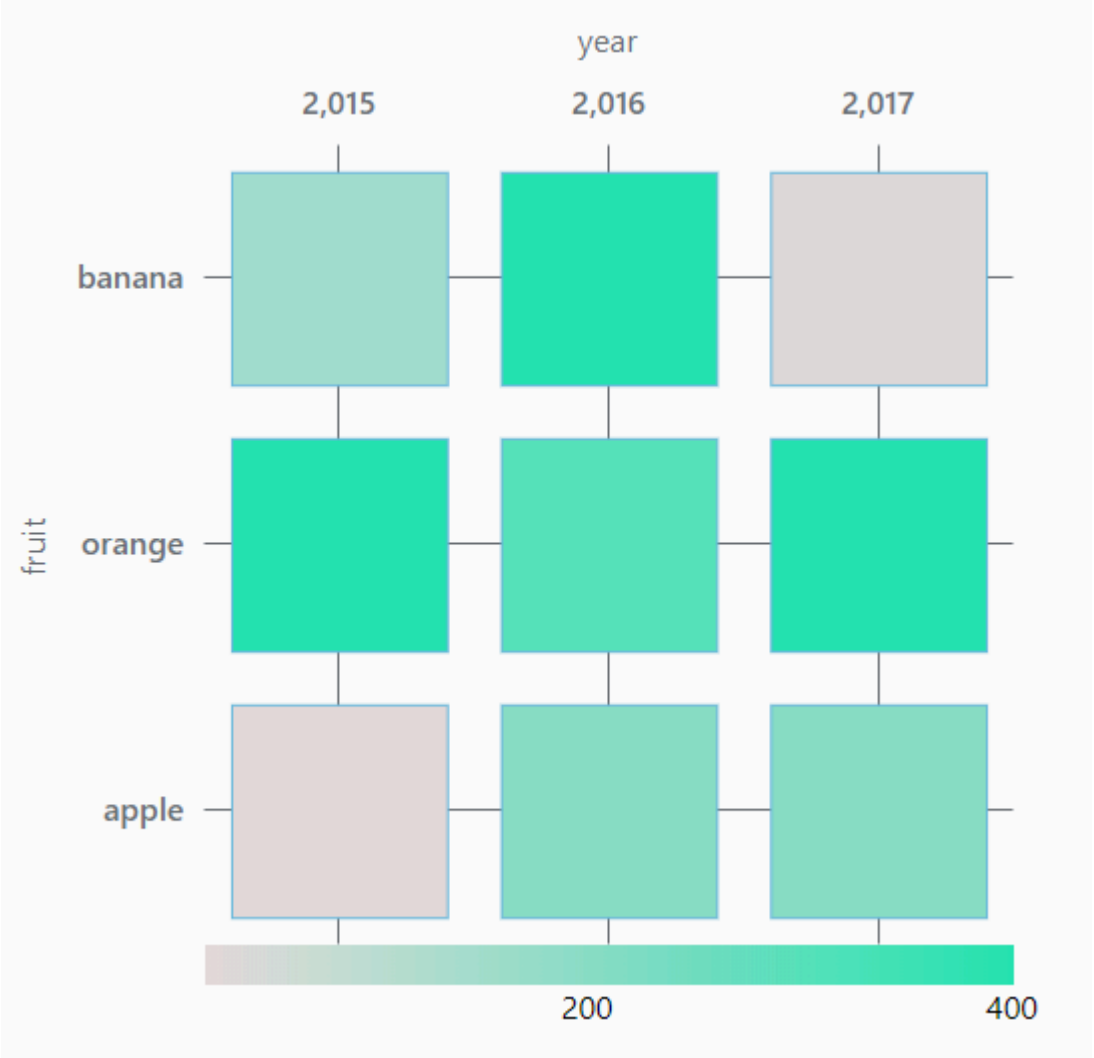
```
        "tensionY": 1,
        "minDistance": 0.5,
        "stroke": {
          "width": 3,
          "opacity": 1,
          "color": "",
          "dashArray": ""
        },
        "fill": {
          "opacity": 0,
          "color": ""
        },
        "bullets": [
          {
            "enabled": true,
            "render": "circle",
            "label": {
              "text": "{value}",
              "position": {
                "dx": 0,
                "dy": 0
              },
              "truncate": true,
              "hideOversized": true
            },
            "fill": {
              "color": "",
              "opacity": 1
            },
            "stroke": {
              "color": "",
              "opacity": 1,
              "width": 1
            },
            "rotation": 0,
            "radius": 5,
            "tooltip": {
              "enabled": true,
              "text": "{name}: [bold]{valueY}[/]",
              "cornerRadius": 3,
              "pointerLength": 4,
              "getFillFromObject": true,
              "background": {
                "color": "",
                "opacity": 1
              }
            },
            "deriveFieldsFromData": {
              "fill": {
```

```
                "color": "",
                "opacity": ""
              },
              "stroke": {
                "color": "",
                "opacity": "",
                "width": ""
              },
              "rotation": ""
            },
            "heatRules": {
              "enabled": false,
              "max": 100,
              "min": 2,
              "dataField": ""
            },
            "width": 10,
            "height": 10
          }
        ]
      }
    }
  ],
  "dataSources": {
    "example": [
      {
        "name": "John",
        "fromDate": "2018-01-01 08:00",
        "toDate": "2018-01-01 10:00",
        "color": "#00FF30"
      },
      {
        "name": "John",
        "fromDate": "2018-01-01 12:00",
        "toDate": "2018-01-01 15:00",
        "color": "#FF0030"
      },
      {
        "name": "John",
        "fromDate": "2018-01-01 15:30",
        "toDate": "2018-01-01 21:30",
        "color": "#00FF30"
      },
      {
        "name": "Jane",
        "fromDate": "2018-01-01 09:00",
        "toDate": "2018-01-01 12:00",
        "color": "#00FF30"
```

```
          },
          {
            "name": "Jane",
            "fromDate": "2018-01-01 13:00",
            "toDate": "2018-01-01 17:00",
            "color": "#FF0030"
          },
          {
            "name": "Peter",
            "fromDate": "2018-01-01 11:00",
            "toDate": "2018-01-01 16:00",
            "color": "#00FF30"
          },
          {
            "name": "Peter",
            "fromDate": "2018-01-01 16:00",
            "toDate": "2018-01-01 19:00",
            "color": "#00FF30"
          }
        ]
      }
    },
    "meta": {
      "name": "XYChart_1"
    },
    "position": {
      "x": 11,
      "height": 375,
      "width": 700
    },
    "custom": {}
  }
]
```

XY Chart Example - Heat Map

This example demonstrates how to configure a Heatmap, where the value of each cross axis is represented as a color along a gradient. This style of chart is largely accomplished by setting the series "render" property to "column", and enabling "heatRules".



The data used in this example is shown in the code block below: The x-axis is set to the "year" key, while the y-axis is set to the "fruit" key. The "dataField" property under "heatRules" is set to the "count" key.

- year - the x-axis
- fruit - the y-axis
- count - determines the color used in the block.

```
[
  {
    "year": 2015,
    "fruit": "apple",
    "count": 20
  },
  {
    "year": 2015,
    "fruit": "orange",
    "count": 400
  },
  {
    "year": 2015,
    "fruit": "banana",
    "count": 150
  },
  {
    "year": 2016,
    "fruit": "apple",
    "count": 200
  },
  {
    "year": 2016,
    "fruit": "orange",
    "count": 300
  },
  {
    "year": 2016,
    "fruit": "banana",
    "count": 400
  },
  {
    "year": 2017,
    "fruit": "apple",
    "count": 200
  },
  {
    "year": 2017,
    "fruit": "orange",
    "count": 400
  },
  {
    "year": 2017,
    "fruit": "banana",
    "count": 30
  }
]
```

Notable Property Configurations

The example requires both an X axis and a Y axis in addition to some series properties.

PROPS Path	Property Description	Value
`xAxes.0.render`	Makes the chart group values in the X Axis, and provides equal padding between each category. Partially responsible for rendering boxes on the chart.	category
`yAxes.0.render`	Makes the chart group values in the Y Axis, and provides equal padding between each category. Partially responsible for rendering boxes on the chart.	category
`series.0.render`	When combined with "category" renders for the X and Y axes, allows the categories boxes to be rendered on the chart.	column
`series.0.column.appearance.heatRules.enabled`	Makes the chart change the color on each category based on dataField key ("count", in our example), applying the min and max colors.	true
`series.0.column.appearance.heatRules.max`	The property represents which color to use for higher values.	#26E3B1

PROPS Path	Property Description	Value
`series.0.column.appearance.heatRules.min`	The property represents which color to use for lower values.	#E5DBDB (pick a color you want to represent low colors)
`series.0.column.appearance.heatRules.dataField`	Determines which key in the underlying data should be used to determine the vibrancy of each block.	count

Example Configuration

The JSON string below can be used to replicate the heatmap example. Simply copy the contents of the code block below and paste it into a container in your Designer.

```
[
  {
    "type": "ia.chart.xy",
    "version": 0,
    "props": {
      "legend": {
        "enabled": false
      },
      "xAxes": [
        {
          "name": "year",
          "label": {
            "enabled": true,
            "text": "year",
            "color": ""
          },
          "visible": true,
          "tooltip": {
            "enabled": true,
            "text": "",
            "cornerRadius": 3,
            "pointerLength": 4,
            "background": {
              "color": "",
              "opacity": 1
            }
          },
          "inversed": false,
          "render": "category",
          "category": {
            "break": {
              "enabled": false,
              "startCategory": "",
              "endCategory": "",
              "size": 0.05
            }
          },
          "date": {
            "baseInterval": {
              "enabled": false,
              "timeUnit": "hour",
              "count": 1,
              "skipEmptyPeriods": false
            },
            "range": {
              "max": "",
              "min": "",
              "useStrict": false
```

```
      },
      "break": {
        "enabled": false,
        "startDate": "",
        "endDate": "",
        "size": 0.05
      },
      "inputFormat": "yyyy-MM-dd kk:mm:ss",
      "format": "M/d"
    },
    "value": {
      "range": {
        "max": "",
        "min": "",
        "useStrict": false
      },
      "logarithmic": false,
      "break": {
        "enabled": false,
        "startValue": 0,
        "endValue": 100,
        "size": 0.05
      },
      "format": "#,###.##"
    },
    "appearance": {
      "opposite": true,
      "inside": false,
      "labels": {
        "color": "",
        "opacity": 1,
        "rotation": 0,
        "verticalCenter": "middle",
        "horizontalCenter": "middle"
      },
      "grid": {
        "color": "",
        "opacity": 1,
        "dashArray": "",
        "minDistance": 60,
        "position": 0.5
      },
      "font": {
        "size": "",
        "weight": 500
      }
    }
  }
],
```

```
    "yAxes": [
      {
        "name": "fruit",
        "label": {
          "enabled": true,
          "text": "fruit",
          "color": ""
        },
        "visible": true,
        "tooltip": {
          "enabled": true,
          "text": "",
          "cornerRadius": 3,
          "pointerLength": 4,
          "background": {
            "color": "",
            "opacity": 1
          }
        },
        "inversed": false,
        "render": "category",
        "category": {
          "break": {
            "enabled": false,
            "startCategory": "",
            "endCategory": "",
            "size": 0.05
          }
        },
        "date": {
          "baseInterval": {
            "enabled": false,
            "timeUnit": "hour",
            "count": 1,
            "skipEmptyPeriods": false
          },
          "range": {
            "max": "",
            "min": "",
            "useStrict": false
          },
          "break": {
            "enabled": false,
            "startDate": "",
            "endDate": "",
            "size": 0.05
          },
          "inputFormat": "yyyy-MM-dd kk:mm:ss",
          "format": "M/d/yyyy HH:mm:ss"
```

```
      },
      "value": {
        "range": {
          "max": "",
          "min": "",
          "useStrict": false
        },
        "logarithmic": false,
        "break": {
          "enabled": false,
          "startValue": 0,
          "endValue": 100,
          "size": 0.05
        },
        "format": "#,###.##"
      },
      "appearance": {
        "opposite": false,
        "inside": false,
        "labels": {
          "color": "",
          "opacity": 1,
          "rotation": 0,
          "verticalCenter": "middle",
          "horizontalCenter": "middle"
        },
        "grid": {
          "color": "",
          "opacity": 1,
          "dashArray": "",
          "minDistance": null,
          "position": 0.5
        },
        "font": {
          "size": "",
          "weight": 500
        }
      }
    }
  ],
  "series": [
    {
      "name": "count",
      "label": {
        "text": "Process Temp"
      },
      "visible": true,
      "hiddenInLegend": false,
      "defaultState": {
```

```
      "visible": true
    },
    "data": {
      "source": "data",
      "x": "year",
      "y": "fruit"
    },
    "xAxis": "year",
    "yAxis": "fruit",
    "zIndex": 0,
    "tooltip": {
      "enabled": true,
      "text": "{name}: [bold]{valueY}[/]",
      "cornerRadius": 3,
      "pointerLength": 4,
      "background": {
        "color": "",
        "opacity": 1
      }
    },
    "render": "column",
    "candlestick": {
      "open": {
        "x": "",
        "y": ""
      },
      "high": {
        "x": "",
        "y": ""
      },
      "low": {
        "x": "",
        "y": ""
      },
      "appearance": {
        "fill": {
          "color": "",
          "opacity": 1
        },
        "stroke": {
          "color": "",
          "opacity": 1,
          "width": 1
        },
        "stacked": false,
        "deriveFieldsFromData": {
          "fill": {
            "color": "",
            "opacity": ""
```

```
        },
        "stroke": {
          "color": "",
          "opacity": "",
          "width": ""
        }
      },
      "heatRules": {
        "enabled": false,
        "max": "",
        "min": "",
        "dataField": ""
      }
    }
  },
  "column": {
    "open": {
      "x": "",
      "y": ""
    },
    "appearance": {
      "fill": {
        "color": "",
        "opacity": 1
      },
      "stroke": {
        "color": "",
        "opacity": 1,
        "width": 1
      },
      "stacked": false,
      "width": null,
      "height": null,
      "deriveFieldsFromData": {
        "fill": {
          "color": "",
          "opacity": ""
        },
        "stroke": {
          "color": "",
          "opacity": "",
          "width": ""
        }
      },
      "heatRules": {
        "enabled": true,
        "max": "#26E3B1",
        "min": "#E5DBDB",
        "dataField": "count"
```

```
      }
    }
  },
  "line": {
    "open": {
      "x": "",
      "y": ""
    },
    "appearance": {
      "connect": true,
      "tensionX": 1,
      "tensionY": 1,
      "minDistance": 0.5,
      "stroke": {
        "width": 3,
        "opacity": 1,
        "color": "",
        "dashArray": ""
      },
      "fill": {
        "opacity": 0,
        "color": ""
      },
      "bullets": [
        {
          "enabled": false,
          "render": "circle",
          "label": {
            "text": "{value}",
            "position": {
              "dx": 0,
              "dy": 0
            },
            "truncate": true,
            "hideOversized": true
          },
          "fill": {
            "color": "",
            "opacity": 1
          },
          "stroke": {
            "color": "",
            "opacity": 1,
            "width": 1
          },
          "rotation": 0,
          "radius": 5,
          "tooltip": {
            "enabled": true,
```

```
            "text": "{name}: [bold]{valueY}[/]",
            "cornerRadius": 3,
            "pointerLength": 4,
            "getFillFromObject": true,
            "background": {
              "color": "",
              "opacity": 1
            }
          },
          "deriveFieldsFromData": {
            "fill": {
              "color": "",
              "opacity": ""
            },
            "stroke": {
              "color": "",
              "opacity": "",
              "width": ""
            },
            "rotation": ""
          },
          "heatRules": {
            "enabled": false,
            "max": 100,
            "min": 2,
            "dataField": ""
          },
          "width": 10,
          "height": 10
        }
      ]
    }
  },
  "stepLine": {
    "open": {
      "x": "",
      "y": ""
    },
    "appearance": {
      "connect": true,
      "tensionX": 1,
      "tensionY": 1,
      "minDistance": 0.5,
      "stroke": {
        "width": 3,
        "opacity": 1,
        "color": "",
        "dashArray": ""
      },
```

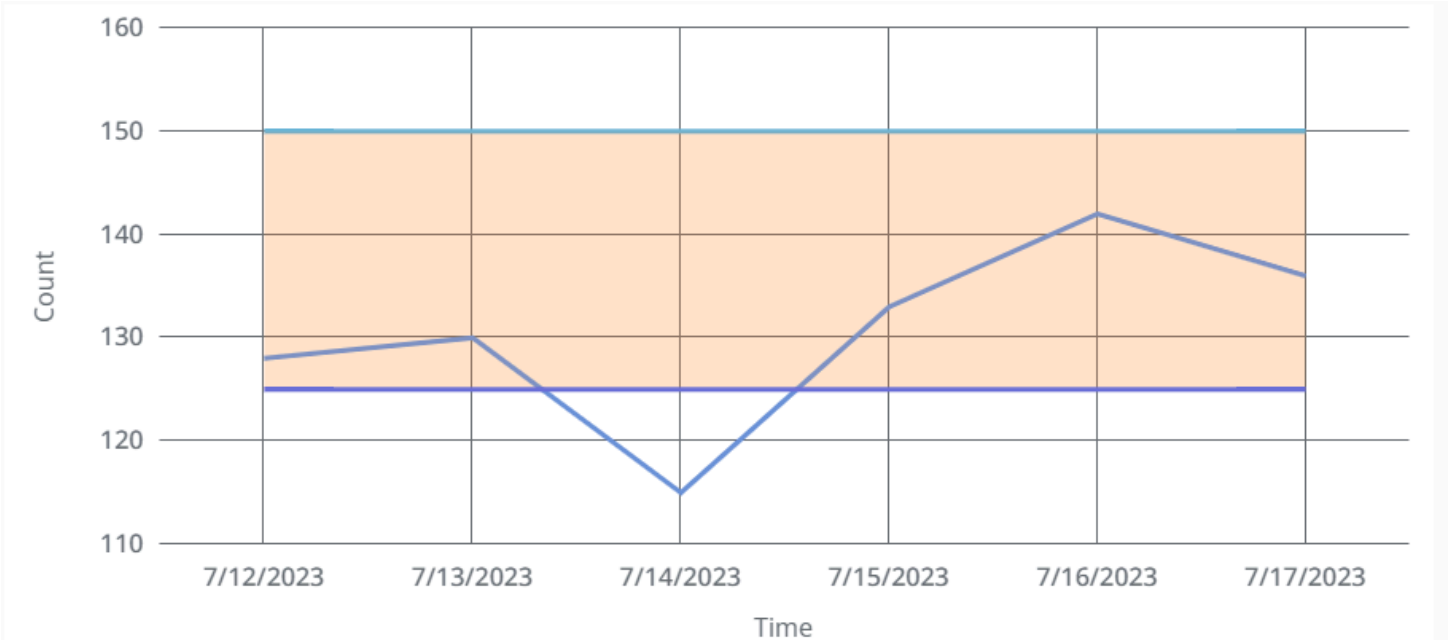
```
        "fill": {
          "opacity": 0,
          "color": ""
        },
        "bullets": [
          {
            "enabled": true,
            "render": "circle",
            "label": {
              "text": "{value}",
              "position": {
                "dx": 0,
                "dy": 0
              },
              "truncate": true,
              "hideOversized": true
            },
            "fill": {
              "color": "",
              "opacity": 1
            },
            "stroke": {
              "color": "",
              "opacity": 1,
              "width": 1
            },
            "rotation": 0,
            "radius": 5,
            "tooltip": {
              "enabled": true,
              "text": "{name}: [bold]{valueY}[/]",
              "cornerRadius": 3,
              "pointerLength": 4,
              "getFillFromObject": true,
              "background": {
                "color": "",
                "opacity": 1
              }
            },
            "deriveFieldsFromData": {
              "fill": {
                "color": "",
                "opacity": ""
              },
              "stroke": {
                "color": "",
                "opacity": "",
                "width": ""
              },
```

```
                "rotation": ""
              },
              "heatRules": {
                "enabled": false,
                "max": 100,
                "min": 2,
                "dataField": ""
              },
              "width": 10,
              "height": 10
            }
          ]
        }
      }
    ],
    "style": {
      "marginRight": "100px",
      "paddingRight": "100px"
    },
    "dataSources": {
      "data": [
        {
          "year": 2015,
          "fruit": "apple",
          "count": 20
        },
        {
          "year": 2015,
          "fruit": "orange",
          "count": 400
        },
        {
          "year": 2015,
          "fruit": "banana",
          "count": 150
        },
        {
          "year": 2016,
          "fruit": "apple",
          "count": 200
        },
        {
          "year": 2016,
          "fruit": "orange",
          "count": 300
        },
        {
          "year": 2016,
```

```
            "fruit": "banana",
            "count": 400
          },
          {
            "year": 2017,
            "fruit": "apple",
            "count": 200
          },
          {
            "year": 2017,
            "fruit": "orange",
            "count": 400
          },
          {
            "year": 2017,
            "fruit": "banana",
            "count": 30
          }
        ]
      }
    },
    "meta": {
      "name": "XYChart"
    },
    "position": {
      "x": 130,
      "y": 103,
      "width": 435,
      "height": 375
    },
    "custom": {}
  }
]
```

XY Chart Example - Line Chart Target Area

This example demonstrates how to utilize the **line.open** property to create a target zone. Target zones can be used to represent acceptable values for the data that is displayed to make it obvious if values are good or bad.



Although, there are many ways to enter data values and high/low limit values for a line chart, the data used in this example is shown in the code block below. Copy the JSON content below and paste it onto the Chart component's **dataSources.example** property.

```
[
  {
    "date": "2023-7-12 07:00:00",
    "actual": 128,
    "highLimit": 150,
    "lowLimit": 125
  },
  {
    "date": "2023-7-13 07:00:00",
    "actual": 130,
    "highLimit": 150,
    "lowLimit": 125
  },
  {
    "date": "2023-7-14 07:00:00",
    "actual": 115,
    "highLimit": 150,
    "lowLimit": 125
  },
  {
    "date": "2023-7-15 07:00:00",
    "actual": 133,
    "highLimit": 150,
    "lowLimit": 125
  },
  {
    "date": "2023-7-16 07:00:00",
    "actual": 142,
    "highLimit": 150,
    "lowLimit": 125
  },
  {
    "date": "2023-7-17 07:00:00",
    "actual": 136,
    "highLimit": 150,
    "lowLimit": 125
  }
]
```


Before creating a target zone, we must first adjust some of the default XY Chart settings to display the **actual** values from our example data source on the line chart:

1. Remove the legend by setting **legend.enabled** to false.
2. Navigate to the xAxis property to set the **xAxis.0.name** and **xAxis.0.label.text** properties. In this example, the name will be **time** and the label.text property will be set to **Time**.

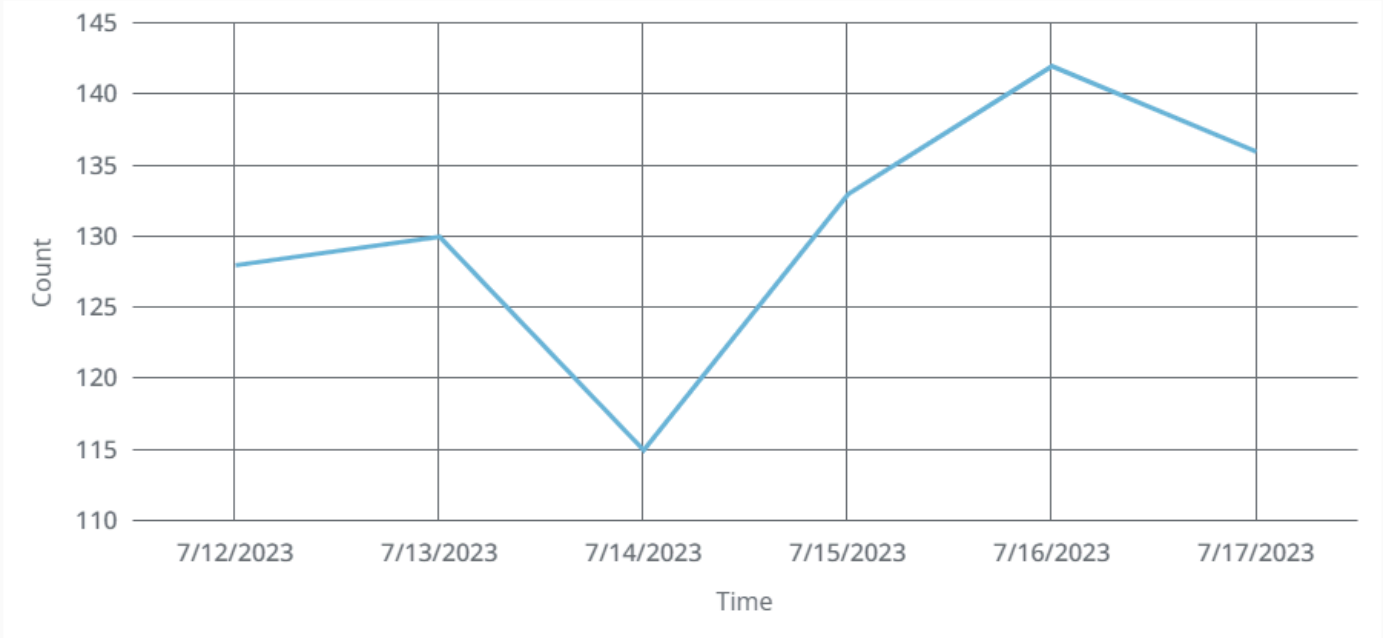
3. Make sure the **xAxes.0.render** property is set to **date** and set your **xAxes.0.date** format if needed. This example uses **date** for the **xAxes.0.date.inputFormat** and **xAxes.0.date.format** properties.

4. Navigate to the **yAxes** property and delete the **yAxes.1** array that is loaded by default.

5. Then, set the **yAxes.0.name** and **label.text** properties. In this example, the name will be **count** and the label.text property will be set to **Count**.

6. Now, set the following **series.0** properties to display the **actual** values:

    - **name**: actual
    - **data.source**: example
    - **data.x**: date
    - **data.y**: actual
    - **xAxis**: time
    - **yAxis**: count



Now, you'll see a line chart that displays the actual values. However, there is still no clear way to tell if these values are good or bad. This will be achieved by creating two more series using the high and low limit data already built into our data source.

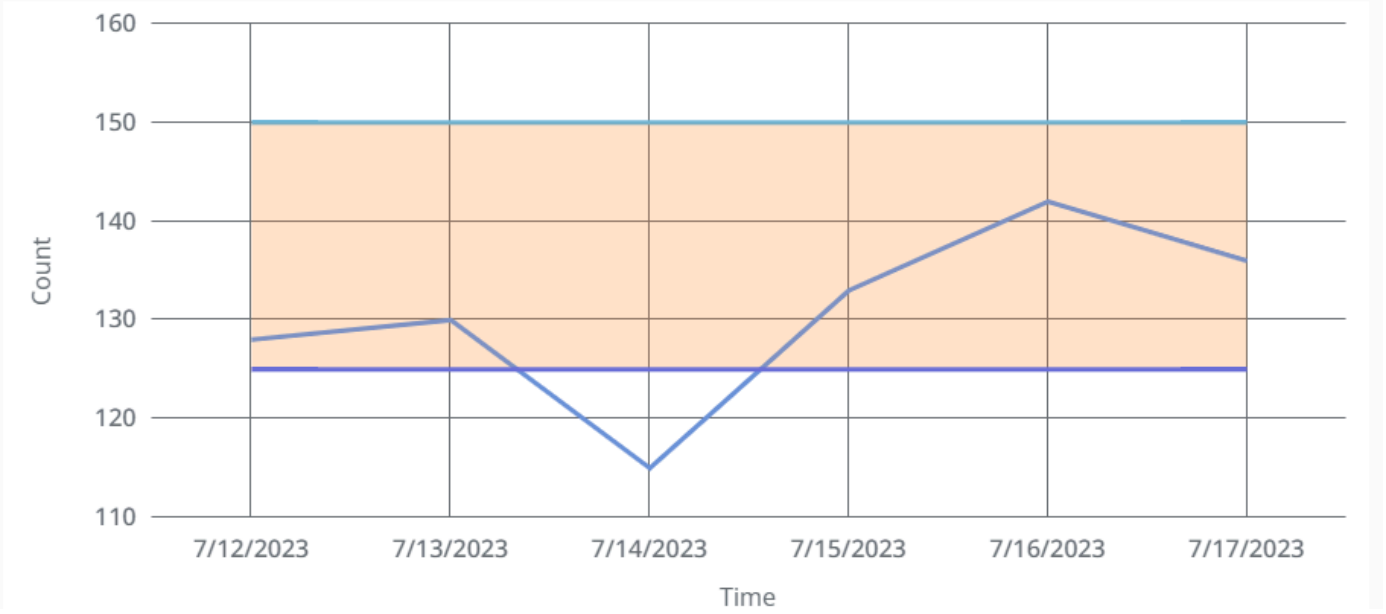
1. Create a new series and set the following **series.1** properties to display the acceptable high limit area.

    - **name**: highLimit
    - **data.source**: example
    - **data.x**: date

- **data.y**: highLimit
- **xAxis**: time
- **yAxis**: count
- **render**: line
- **line.open.y**: lowLimit
- **line.appearance.fill.color**: #FF8C00
- **line.appearance.fill.opacity**: 0.125

2. Duplicate series.1 and change the following **series.2** properties to display the acceptable low limit area.

    - **name**: lowLimit
    - **data.y**: lowLimit
    - **line.open.y**: highLimit



With the high and low limit areas displayed, it is now easy for any user to see that all counts were within the acceptable range, except for on 7/14/2023.

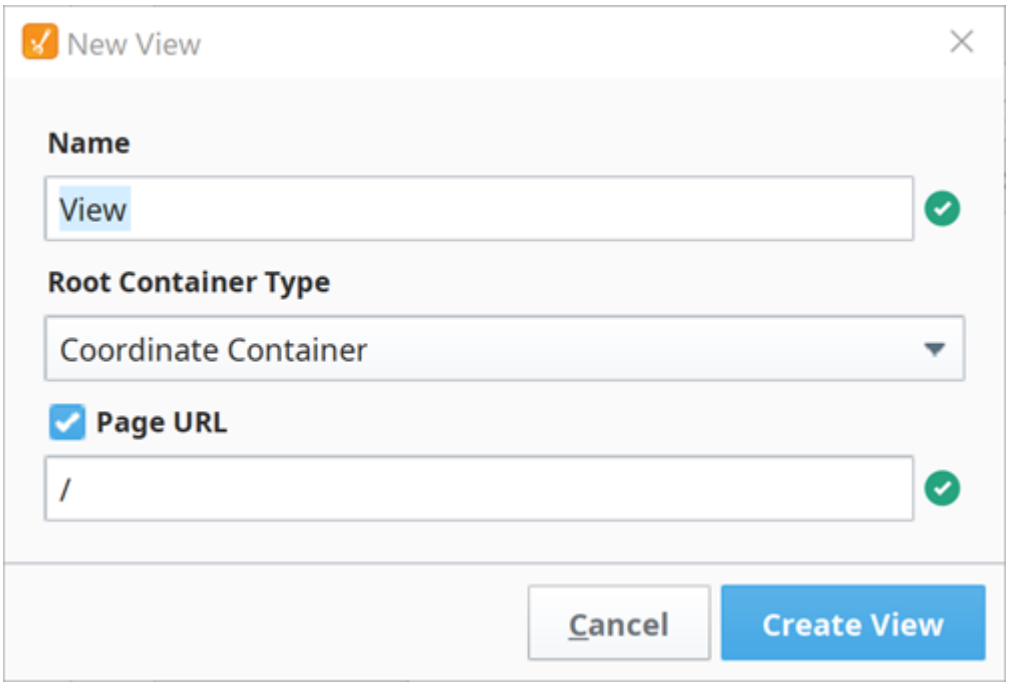
Perspective - Container Palette

Container Components

Container components provide a way of laying out and organizing Perspective components within a view. The different container types support different layout strategies. As the root level of a view, containers are essential to creating responsive applications with the ability to display information across a wide variety of screen sizes and orientations. The container type is selected when creating a view under the Root Container Type field. Once the view is created, the root container type cannot be changed.

The following is a complete list of container components, and a link pointing to a page with the component's description, properties, and an example of how to configure it.

The Coordinate container is the default container type when creating a new view in a project. A different container type can be selected using the dropdown arrow to display the full list of container component options.

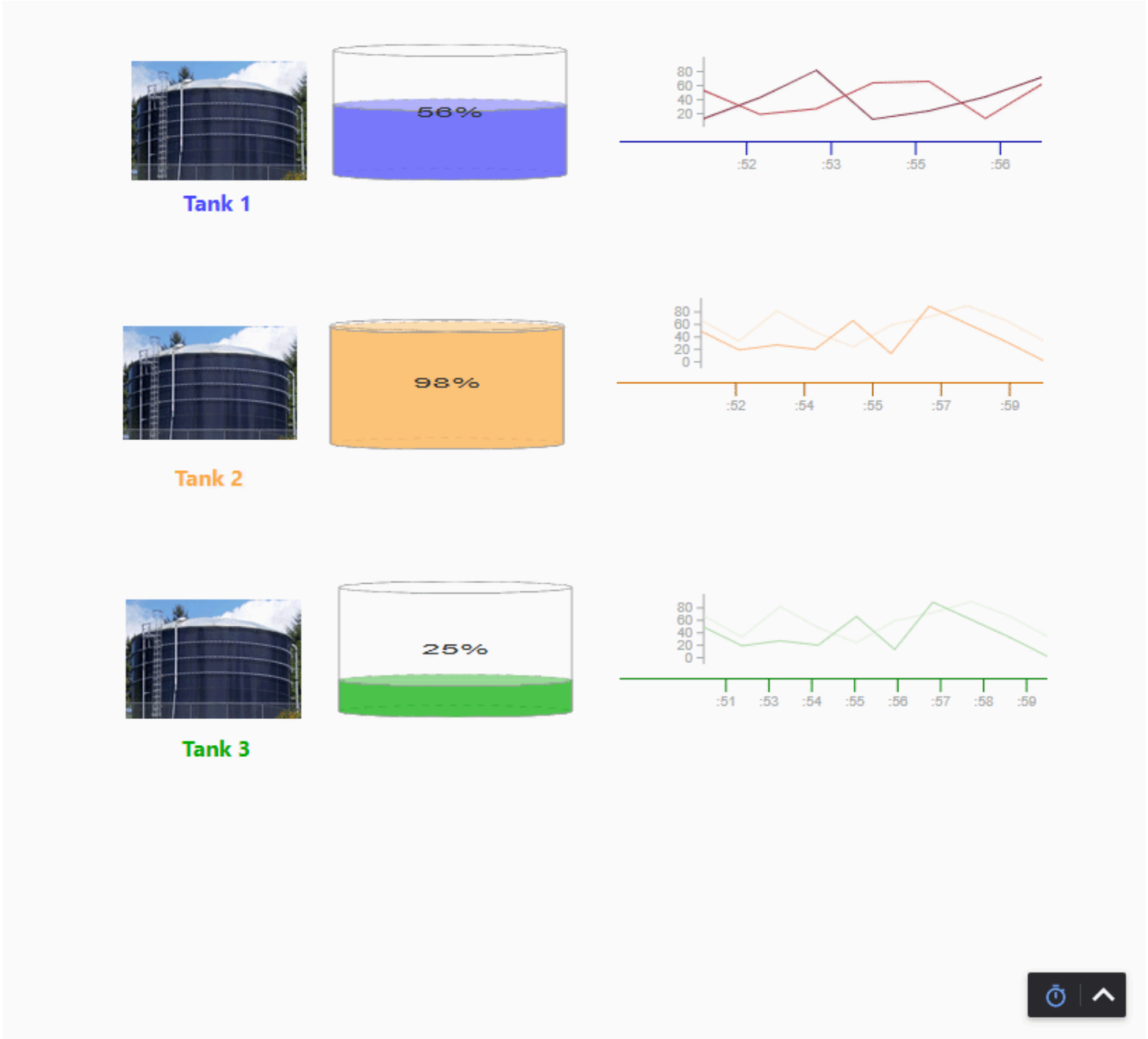


After the initial view is created, all new views will default to the last used container type selected in the Root Container Type field. For example, if the last view you created used the Flex Container, the next time you create a new view the Root Container Type will default to Flex Container instead of Coordinate Container.

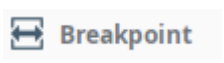
The following is a complete list of container components, and a link pointing to a page containing the component's description, properties, and an example of how to configure it.

- Breakpoint Container
- Column Container
- Coordinate Container
- Flex Container
- Tab Container
- Split Container

Perspective - Breakpoint Container



Component Palette Icon:



Description

The Breakpoint container consists of a single breakpoint, with two child views. In other words, using a Breakpoint container offers you a layout with the opportunity to create two different views that are

shown at two distinct ranges of layout widths.

This allows for a very simple responsive design that removes one container and replaces it with another when the viewport width is changed. With the Breakpoint container, you can define completely different content to render for each child. This is in contrast to a container such as the column container, where the content for each breakpoint is identical but the layout of it changes according to screen size.

Breakpoint containers are ideal to use in cases where you want completely different components available between a mobile user or a desktop user, since the components in each breakpoint are completely separate instances. Thus, if you're attempting to make a responsive view, and a Flex container isn't quite giving the desired result, a Breakpoint container can be used to switch to a completely different container or view.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
breakpoint	Width (in pixels) breakpoint declarations for child layouts. When the container is sized below minWidth, child position rules will fall back to the next set breakpoint rules.	value: numeric
currentBreakpoint	**New in 8.3.4** A read-only property that displays the container's current size according to its breakpoint. Values include **small** and **large**.	value: string
determinant	Choose between the height or width dimension that the Breakpoint container will use to break. Default is width.	value: string
style	Use Style to customize the visual style of the component. The **Style menu** contains all the tools for modifying text, background, margins, and borders. You can also specify a **style class**.	object

Child Component Position Properties

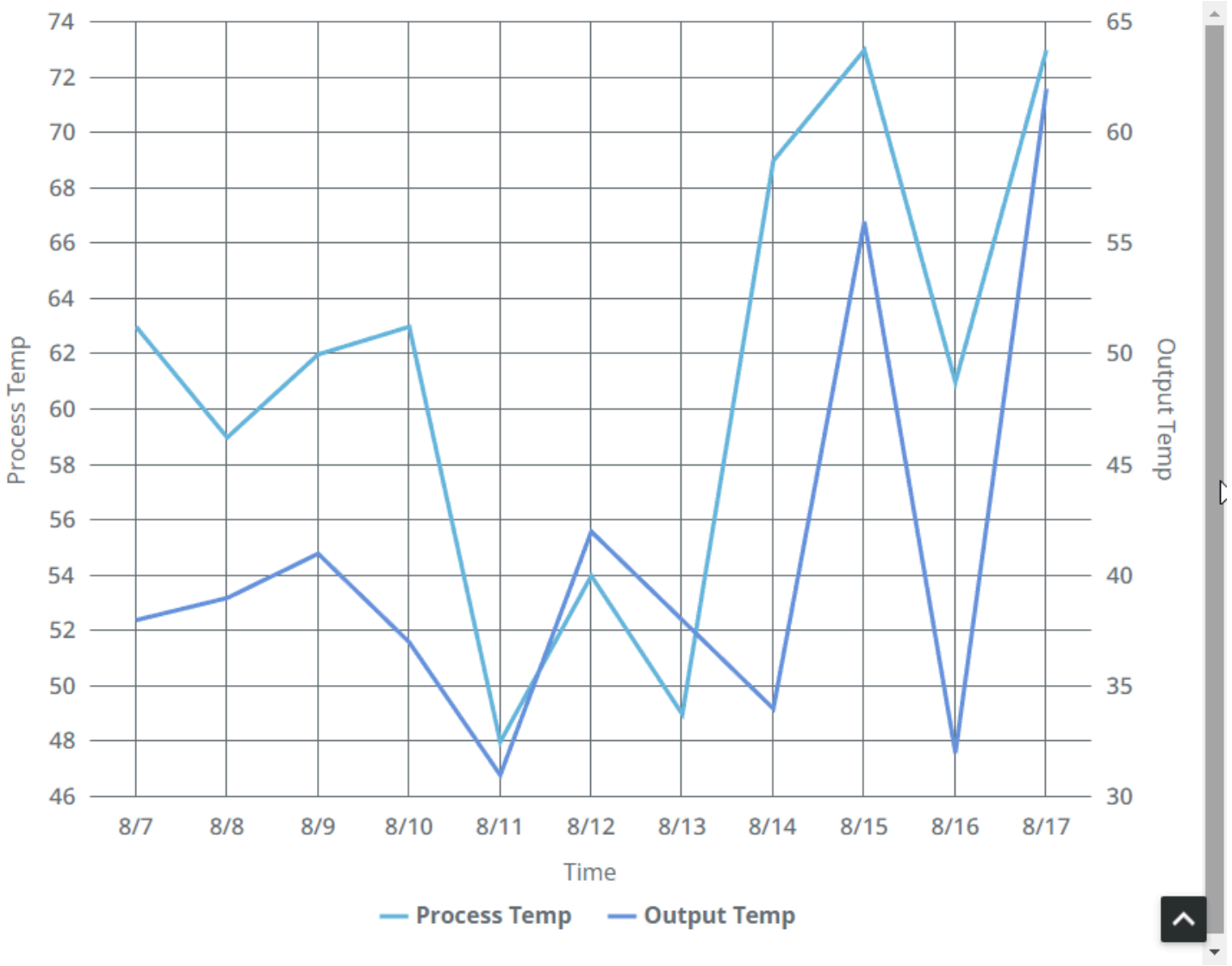
When a component is placed inside of a Breakpoint container, it will inherit the position property listed below.

Name	Description	Property Type
size	Indicates which child the component located in. Expected values are **small** and **large**.	value: string

Scripting

See the **Perspective - Breakpoint Container Scripting** page for the full list of scripting functions available for this component.

Nested Breakpoint Containers Example



In this example, we have a Breakpoint container nested inside a root Breakpoint container. The setup is as follows:

Containers

- The nested Breakpoint container is used when the root Breakpoint container's size is less than 800 pixels.
- The nested Breakpoint container will change when its size is less than 700 pixels.

Components

- When the page is greater than 800 pixels, an XY Chart is displayed.
- When the page is between 700 and 800 pixels, an empty Power Chart is shown.
- When the page is less than 700 pixels, an Icon is displayed.

The relevant properties for this example are displayed below:

Root Breakpoint Container Properties

Property	Value
Breakpoint	800px

Nested Breakpoint Container Properties

Property	Value
Breakpoint	700px

Perspective - Breakpoint Container Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Breakpoint Container** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Functions

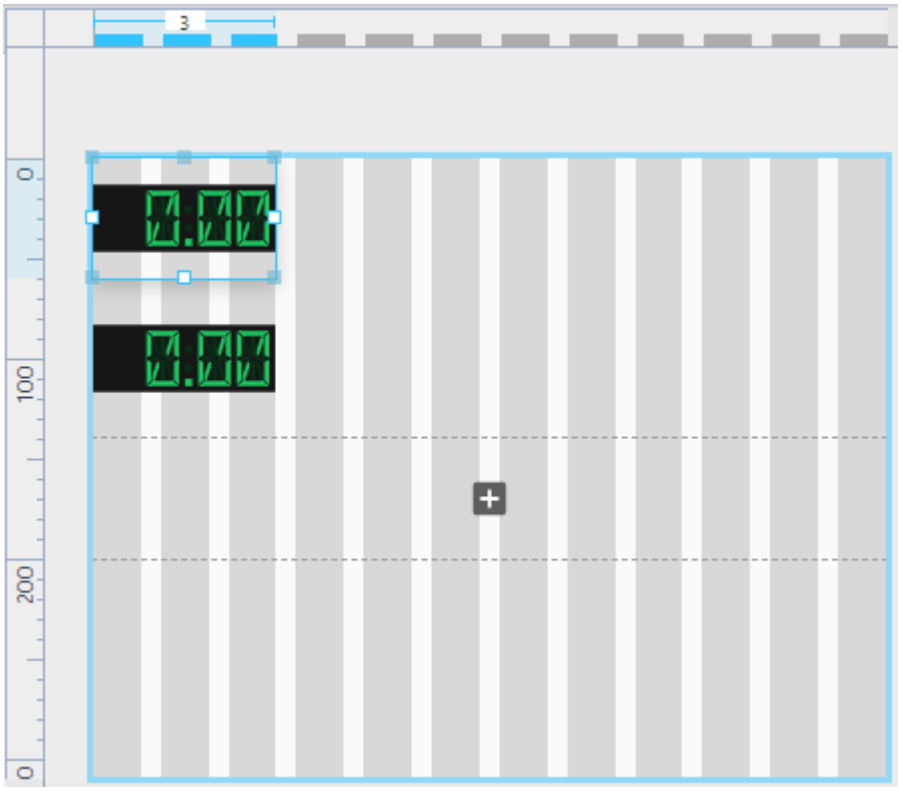
.getChildren()

- Description
  - Returns an ArrayList, which contains references to all components inside of the container.
- Parameters
  - None
- Return
  - Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

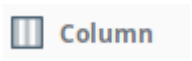
Extension Functions

This component does not have extension functions associated with it.

Perspective - Column Container



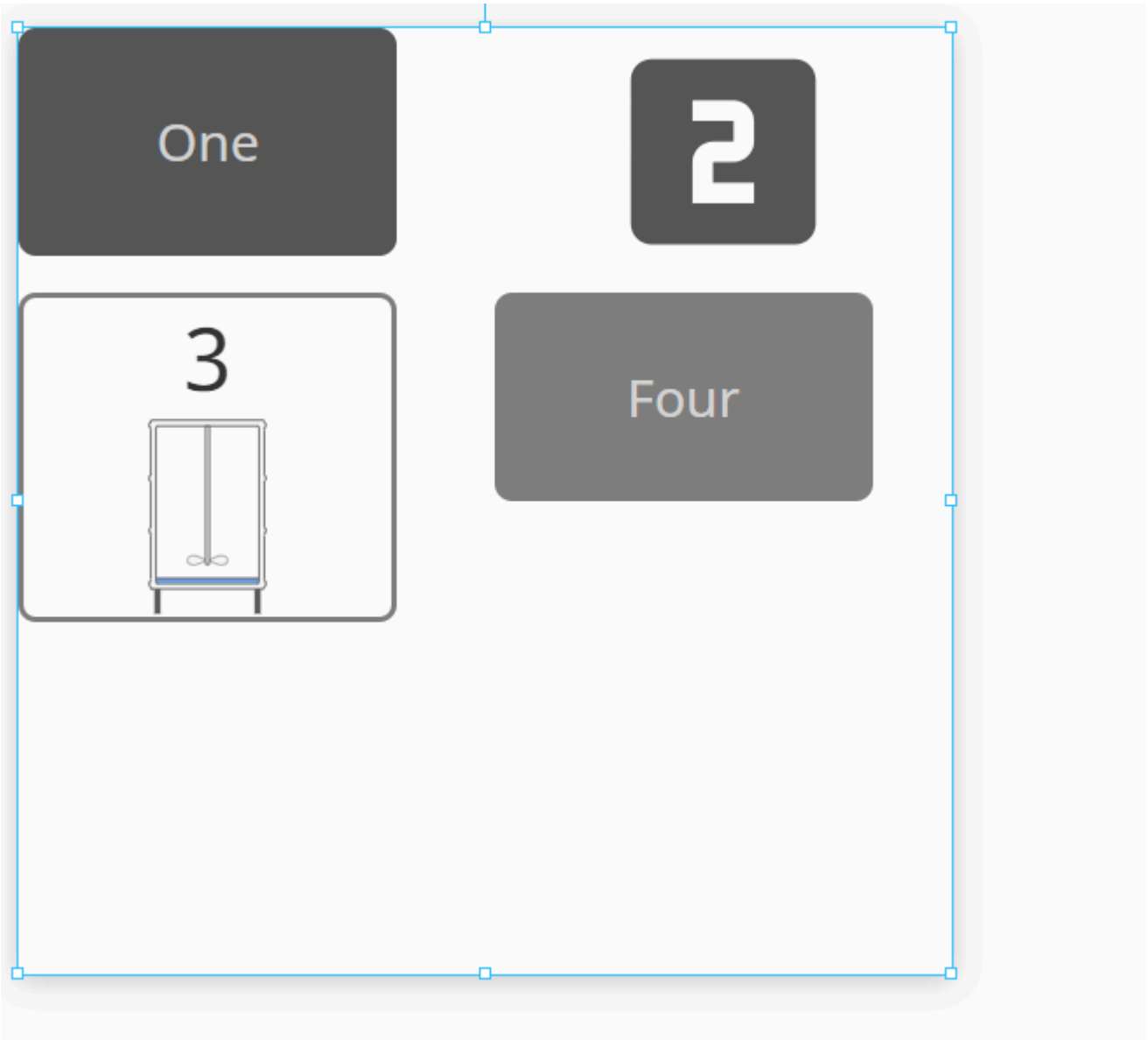
Component Palette Icon:



Description

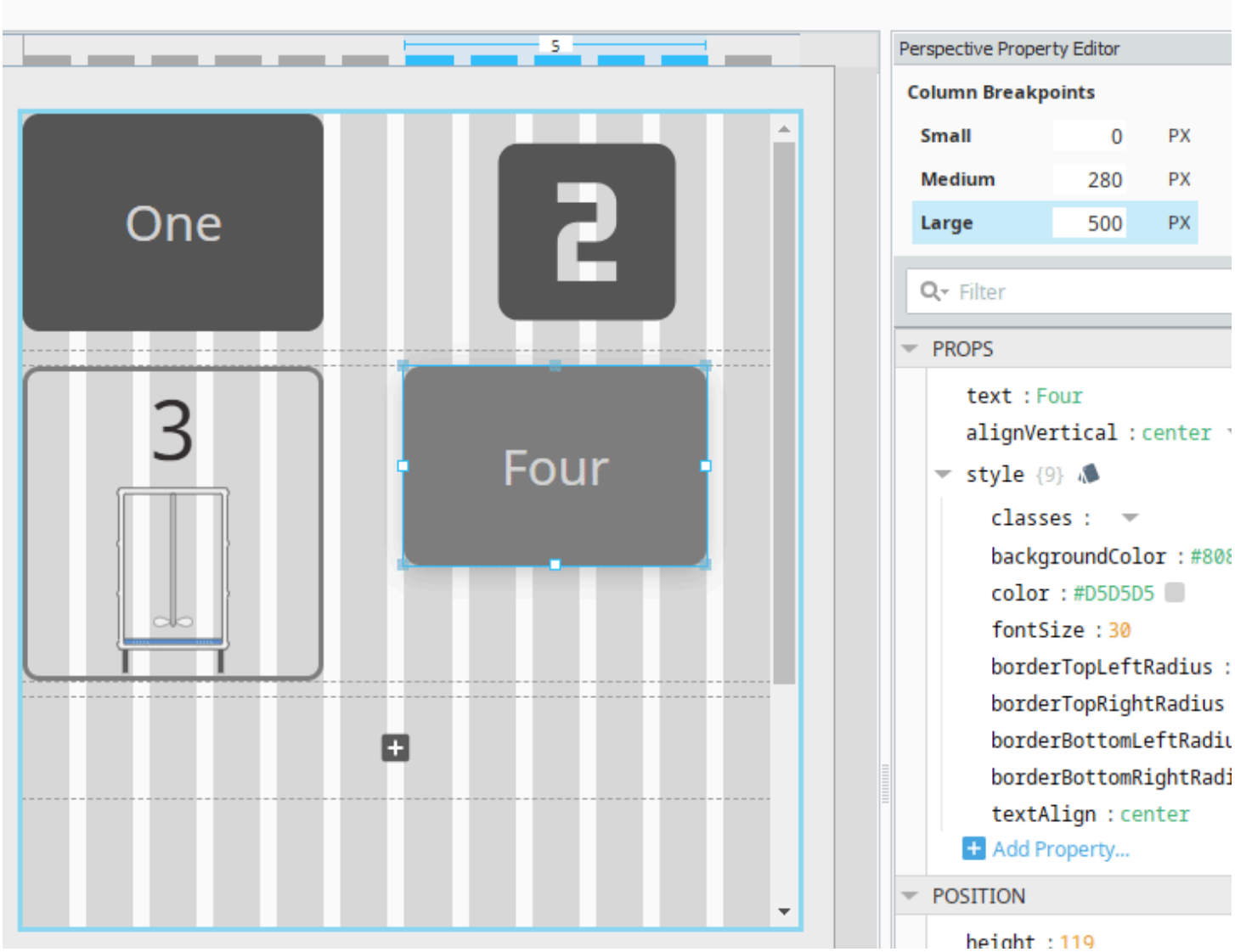
The Column Container is a 12 column grid layout, where components can be organized into columns which alter their layout depending on screen viewport size. The container features three different breakpoints that it can switch between as the viewport changes. Each component inside the container stores positioning and sizing values per breakpoint, allowing you to determine the location and size of each component for each breakpoint.

A column container provides a way to create a single set of components that can be arranged up to three ways depending on the width of the container. Components are able change how many columns they span when switching between breakpoints, as well as change row and column index altogether. In the GIF below, the components are resizing and repositioning as the viewport on the container changes.



User Interface

While deep selected, the Perspective Property Editor will show a **Column Breakpoints** interface. Clicking on **Small**, **Medium**, or **Large** will change the container workspace, showing you the current configurations for all child components in the container. When a container workspace is selected, you can freely move and resize components for the selected breakpoint, allowing you to determine their placement when the selected breakpoint is active in a session.



Note that switching between breakpoints in the designer does not resize the container. The pixel dimensions represent a minimum width for the viewport, meaning that the container will switch to the breakpoint when the minimum width is satisfied during runtime. For example, if Medium is set to 280px and Large is 500px, then the container will show the medium breakpoint when the viewport is between 280 and 499 pixels in width.

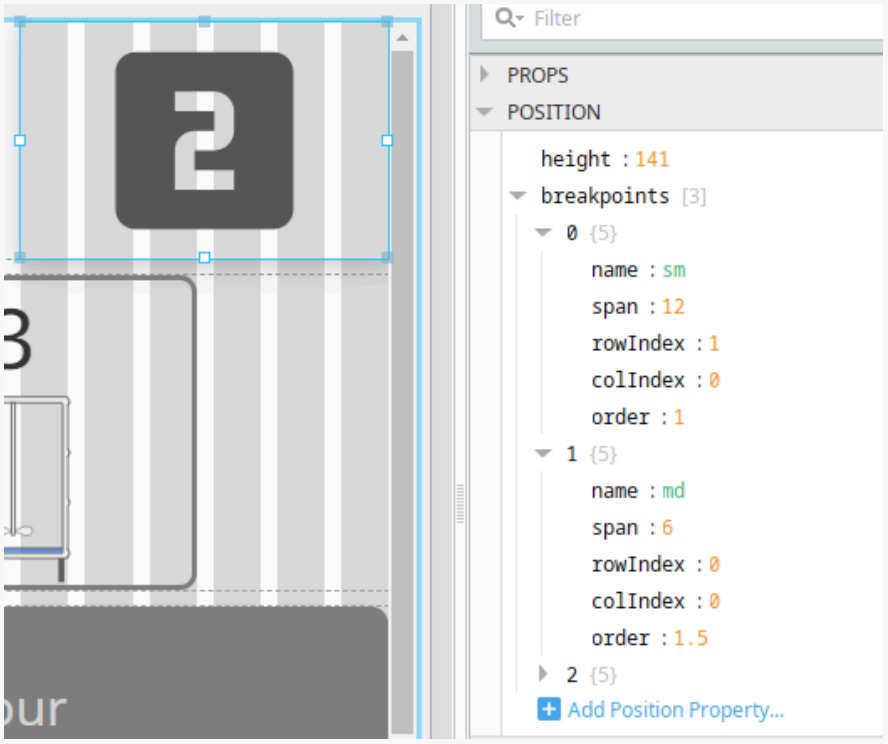
Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Property	Description	Property Type
breakpoints	Width breakpoint declarations for child layouts. When the container is sized below minWidth, child position rules will fall back to the next set breakpoint rules. • **name**: Name of the breakpoint with a string property type. • **minWidth**: Minimum width for this breakpoint with a numeric property type.	array
currentBreakpoint	**New in 8.3.4** A read-only property that displays the container's current size according to its breakpoint. Values include: • sm (small) • md (medium) • lg (large)	value: string
gutters	Settings for the gutters, which are the amount of space, in pixels, to place between child components. • **vertical**: Vertical gutter setting between child components with a numeric property type. • **horizontal**: Horizontal gutter setting between child components with a numeric property type.	object
style	Use Style to customize the visual style of the component. The **Style menu** contains all the tools for modifying text, background, margins, and borders. You can also specify a **style class**.	object

Child Component Position Properties

When a component is placed inside of a Column container, it will inherit the position property listed below.

Property	Description	Property Type
height	Specifies the horizontal positioning of the component in pixels.	value: numeric
breakpoints	An array of objects, where each object represents a different breakpoint. In the image below, a component in a column container is selected. Each element under **breakpoints** contains positioning information that will be used when the container switches to the named breakpoint. 	array

Breakpoints Child Properties

Property	Description	Property Type
name	Name of a breakpoint defined in container. If this matches the currently applied breakpoint, these rules determine child layout. Options are: • sm (small) • md (medium) • lg (large)	value: string
span	Number of columns the child's width will span.	value: numeric
rowIndex	Row index (starting from 0) in which to place child. Children may wrap lines within a row. Children in separate rows don't affect each other's layout.	value: numeric
colIndex	Column number upon which the child's span should begin unless forced to wrap.	value: numeric
order	Where component is places among its siblings within its row. Ordering is independent per row.	value: numeric

Scripting

See the **Perspective - Column Container Scripting** page for the full list of scripting functions available for this component.

Perspective - Column Container Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Column Container** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Functions

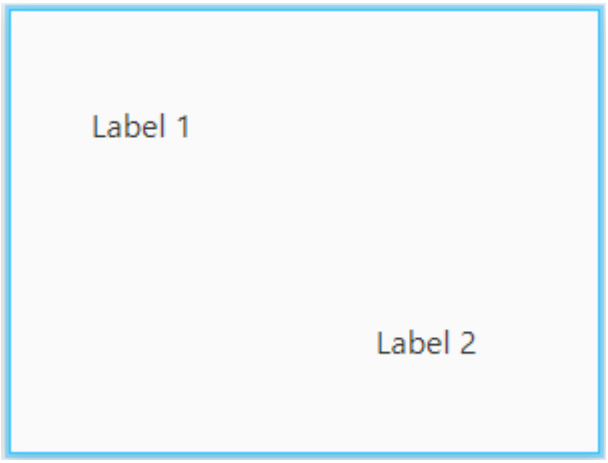
.getChildren()

- Description
  - Returns an ArrayList, which contains references to all components inside of the container.
- Parameters
  - None
- Return
  - Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

Extension Functions

This component does not have extension functions associated with it.

Perspective - Coordinate Container



Component Palette Icon:



Description

The Coordinate Container makes a component's size and location relative to its parent's size and location. Components can be fixed size, or optionally grow/shrink proportionally when the view is stretched.

Coordinate Containers are ideal to use in cases where you need components to overlap each other, such as adding a component on top of another (z-axis) to act as an overlay. They're also useful in cases where you do **not** want components within to resize - for example, building a diagram where each element is a separate component.

Components placed in coordinate containers can be rotated. The Rotate property has been moved to the Position Properties section of the Perspective Property Editor. For more information, see **Working with Perspective Components**.

The Coordinate Container component has two pre-configured **variants**:

- Fixed - Child layouts will be in fixed coordinate space.
- Percent - Child layout will be stretched to different size containers.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
mode	Whether child layouts should always be in fixed coordinate space, or stretched relative to different container sizes: fixed or percent. Fixed mode uses absolute units, which reduces the amount of resizing of components within the container when launched across different display sizes. Percent mode uses relative units, which means child components will be able to resize appropriately when launched on different sized displays.	value: string
aspectRatio	Only applied in percent mode. Optional dimensions, in x:y format to apply to maintain container aspect ratio for different sizes. Empty string (or non x:y input) will disable this mode.	value: string
style	Use Style to customize the visual style of the component. The **Style menu** contains all the tools for modifying text, background, margins, and borders. You can also specify a **style class**.	object

Child Component Position Properties

When a component is placed inside of a coordinate container, it will inherit the position properties listed below. The values for x, y, width, and height will differ based on the mode of the coordinate container.

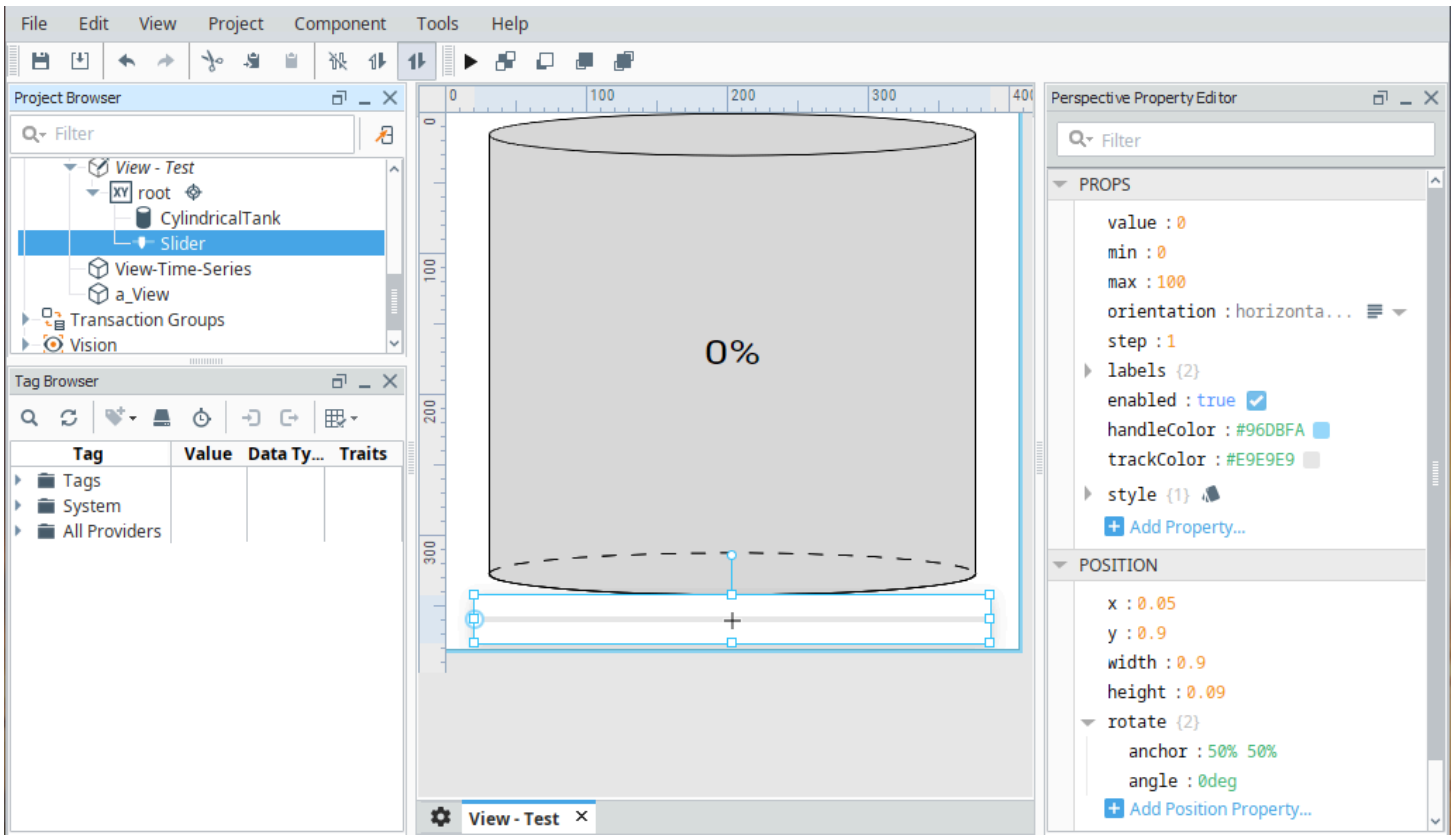
- fixed: Values represent absolute lengths. Example, an x value of 100px means the left edge of the component starts 100 pixels from the left edge of the container.
- percent: Values are in relative lengths. Example, say a component has an x value of 0.25. In this scenario, the left edge of the container would be 0, the right edge would be 1. So 0.25 would be 25% distance from the left edge of the container.

Name	Description	Property Type
x	Specifies the horizontal positioning of the component in pixels.	value: numeric
y	Specifies the vertical positioning of the component in pixels.	value: numeric
width	Specifies the width of the component in pixels.	value: numeric
height	Specifies the height of the component in pixels.	value: numeric
rotate	Setting that sets the anchor and angle of rotation for the component. • **anchor**: The point around which the rotation happens. Either as an `{x:number, y:number}` object where x and y represent percentages such that `{x:0, y:0}` represents the (0%, 0%) or top-left corner of the component, or as a valid CSS transform-origin string. Can be a string or object. • **angle**: How much to rotate the component. Valid values include numbers (as degrees), and valid CSS angle strings such as '45deg', '2rad', '0.5turn', etc. Can be numeric or string	object

Scripting

See the **Perspective - Coordinate Container Scripting** page for the full list of scripting functions available for this component.

Example



In this example, we have a Coordinate container with a Cylindrical Tank component and a Slider component. We've set the container property to percent so that the components will grow and shrink with the container size.

Container properties

Property	Value
props.mode	percent

Cylindrical Tank properties

Property	Value
position.x	0.05
position.y	0
position.width	0.9
position.height	0.9

Slider properties

Property	Value
position.x	0.05
position.y	0.9
position.width	0.9
position.height	0.09

Perspective - Coordinate Container Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Coordinate Container** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onPipeClicked

This event will trigger when a pipe in this container is clicked.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.event

- Object Path
  - event.event
- Type
  - Dictionary
- Description
  - The event object generated from the mouse click.

event.pipeIndex

- Object Path
  - event.pipeIndex
- Type
  - Integer or float
- Description
  - The array index of the pipe within props.pipes that was clicked.

event.pipeName

- Object Path
  - event.pipeName
- Type
  - String
- Description
  - The name of the pipe that was clicked.

Component Functions

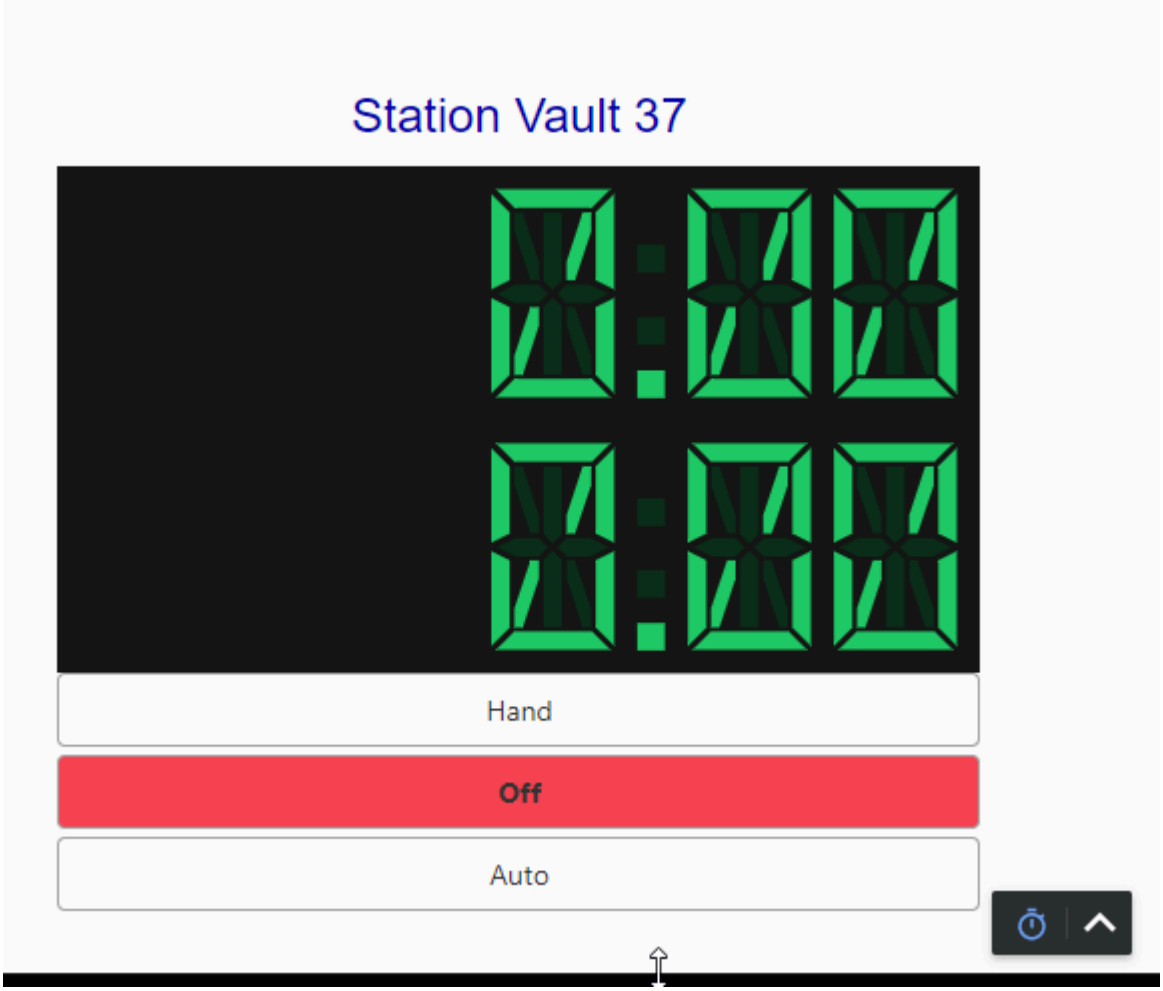
.getChildren()

- Description
  - Returns an ArrayList, which contains references to all components inside of the container.
- Parameters
  - None
- Return
  - Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

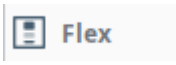
Extension Functions

This component does not have extension functions associated with it.

Perspective - Flex Container



Component Palette Icon:



Description

The Flex container provides an efficient way to lay out, align, and distribute space among components in the container particularly when their size is unknown or dynamic. The Flex Container can alter a component's width and height to best fill the available space to accommodate all types of devices and screen sizes. It expands components to fill available free space, or shrinks them to prevent overflow.

The container works off of two axes: main axis and cross axis. Components will fill out the container along the main axis. Additional alignment and justification of the components are determined by the alignItems, alignContent, and justify properties.

The Flex container is based off the **CSS flexbox**.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Property	Description	Property Type
direction	Direction of the child layout. Options are row, row-reverse, column, column-reverse.	value: string
wrap	Whether the container should allow children to wrap to the next line if space has run out. Options are nowrap, wrap, wrap-reverse.	value: string
justify	Adjusts placement of children along the main axis when there is extra space, which may be used to fill areas before, after, or in-between: flex-start, flex-end, center, space-between, space-around, space-evenly.	value: string
alignItems	Adjusts placement of children along the cross axis when there is extra space: flex-start, flex-end, center, baseline, stretch.	value: string
alignContent	Adjusts alignment of wrapped content when there is free space in the cross axis: flex-start, flex-end, center, space-between, space-around, stretch.	value: string
style	Use Style to customize the visual style of the component. The **Style menu** contains all the tools for modifying text, background, margins, and borders. You can also specify a **style class**.	object

Child Component Position Properties

When a component is placed inside of a flex container, it will inherit the position properties listed below.

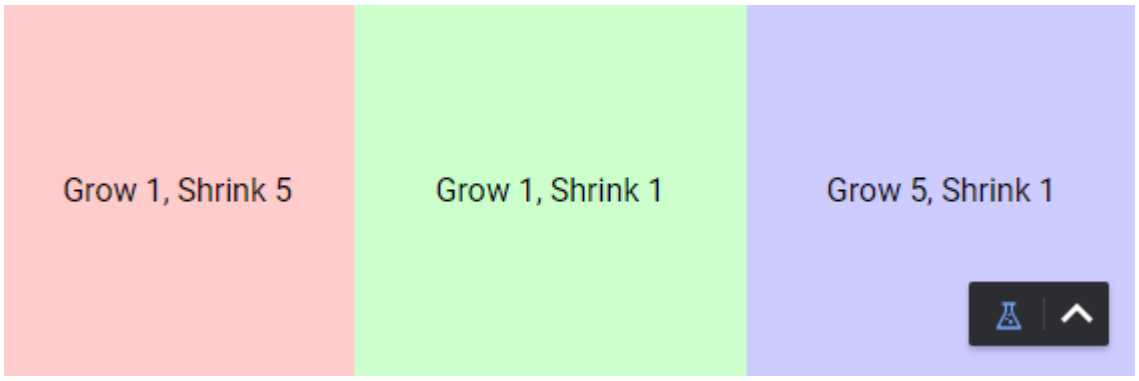
Property	Description	Property Type
grow	Ability to grow in the **direction** dimension as needed, relative to siblings. If space is available and grow is not zero, it may stretch, depending on sibling rules. This value is relative to other components, meaning that two components with the same grow value will grow at the same rate.	value: numeric
shrink	Ability to shrink in **direction** dimension as needed, relative to siblings. If space is available and grow is not zero, it may stretch, depending on sibling rules. This value is relative to other components, meaning that two components with the same shrink value will shrink at the same rate.	value: numeric
basis	Space filled by component by default, before **grow**, **shrink**, and sibling considerations are evaluated. This is the component's base width when the direction property is set to **row**, and it is the component's base height when the direction property is set to **column**.	value: numeric
display	Determines if the component will be displayed in the container or not. Components that are not displayed won't just be invisible, but will actually be removed from the Flex container, readjusting all other components to fit.	value: boolean

User Interface

See the **Flex Container User Interface** page for more information on additional functionality available when the Flex container is deep selected.

Growing and Shrinking

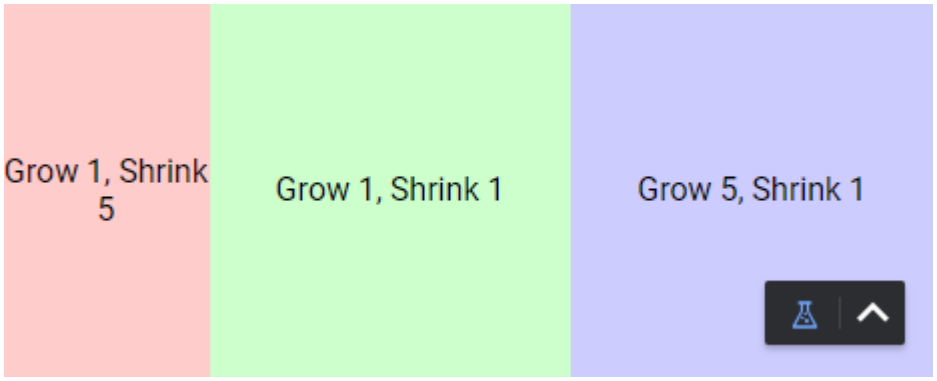
The Flex container's **grow** and **shrink** properties allow a great deal of control over how different components expand and shrink. To demonstrate, here are three components nested inside a Flex container, with a direction of row:



All three components have the same basis, so at this session width, they all have the same size. However, the blue component has a much larger grow value than the other two components. So, when we increase the length of the Flex container along its direction property:



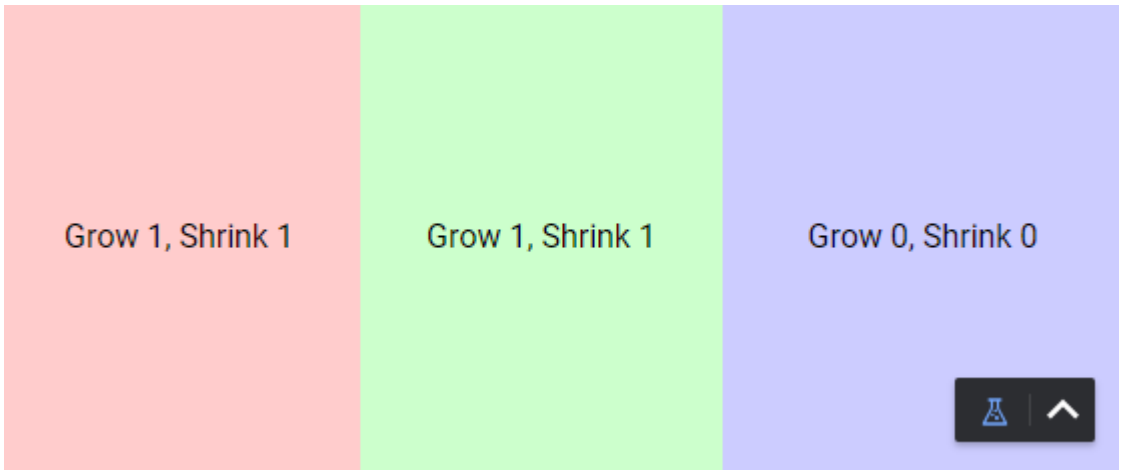
The blue component does most of the stretching. Specifically, since the sum of all grow properties is 1 + 1 + 5 = 7, and the blue component has a grow property of 5, for every 7 pixels the Flex container grows, the blue component will grow by 5 pixels. Now let's try shrinking the container, noting that the red component has a shrink value of 5:



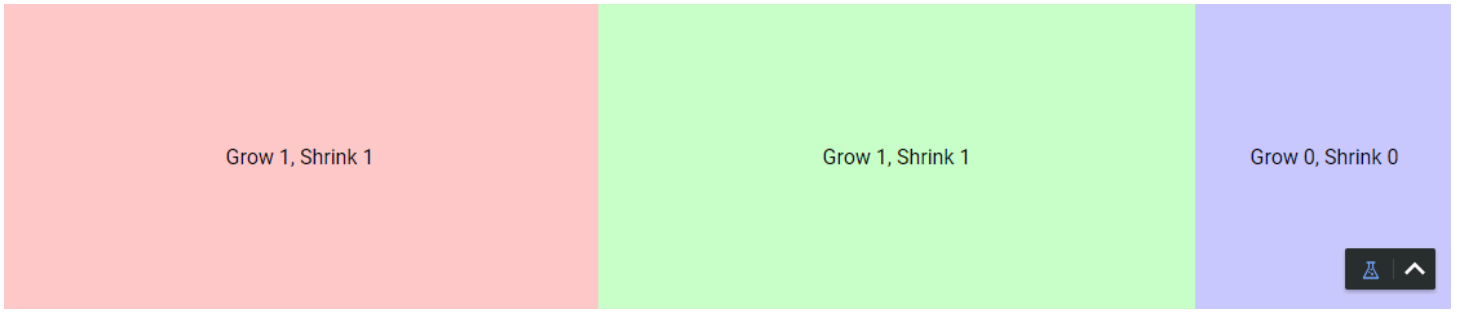
As you can see, a larger shrink value will make the container shrink more. For every 7 pixels the Flex container loses, the red component will lose 5 pixels.

Static Widths

Now let's try the same example again, but with some different grow and shrink values:

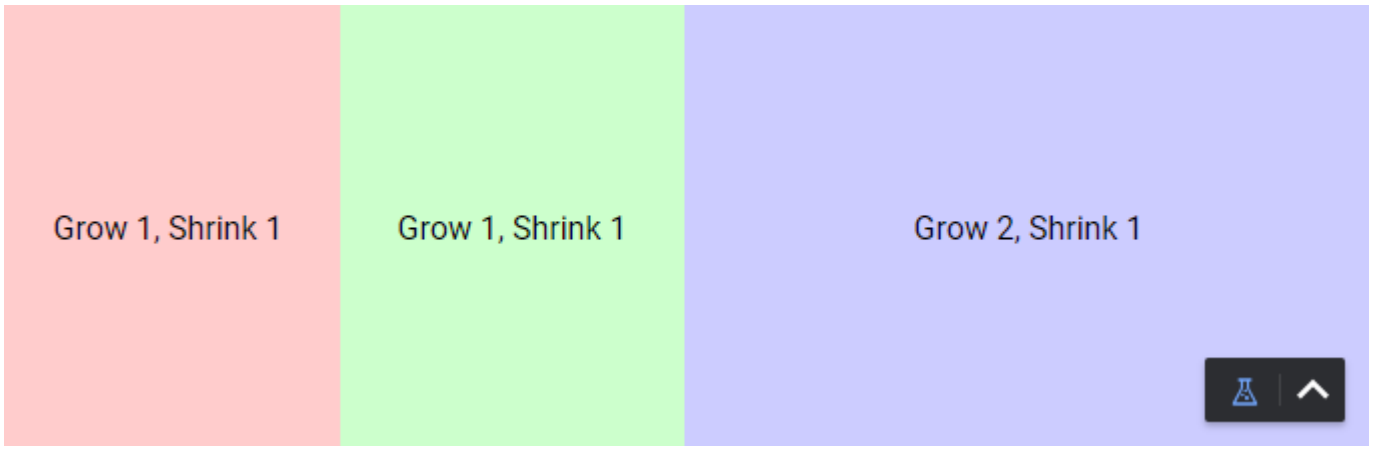


We've given the red and green components identical values, so they should stretch and shrink at the same rate. Meanwhile, the blue component's grow and shrink values are both 0, so when we make the Flex container wider, it stays the same size:

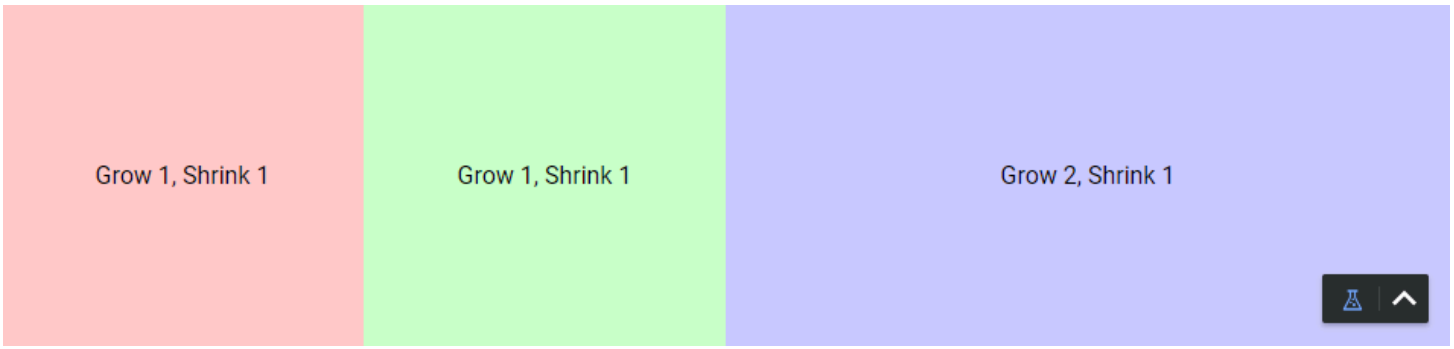


Even Scaling

Now let's say we start off with a blue component twice the **basis** of the others:



In order to maintain this ratio as the width of the container increases, it must have a **grow** value twice that of the others:

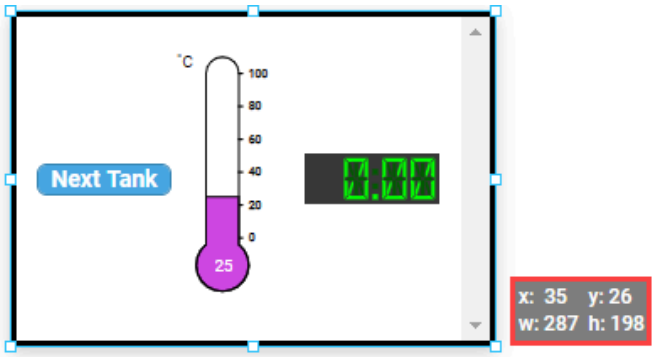


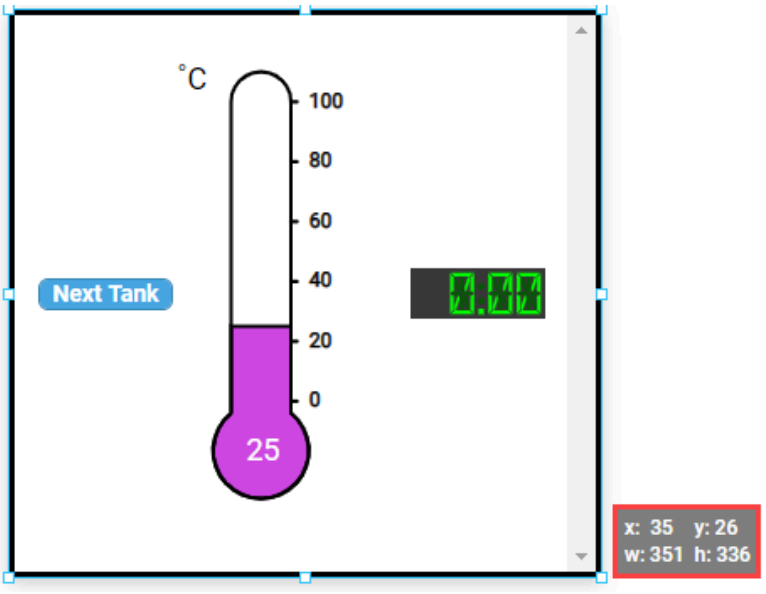
Scripting

See the **Perspective - Flex Container Scripting** page for the full list of scripting functions available for this component.

Example

In the following example, we have three components inside a Flex container: Button, Thermometer, and LED Display. The position properties for the Button and LED Display are set so that their size will not change if the Flex container is resized. However, the Thermometer will shrink or grow depending on the Flex container size.





The following properties are set for the Flex Container:

Property	Value
props.direction	row
props.wrap	nowrap
props.justify	space-between
props.alignItems	center
props.alignContent	flex-start

The following properties are set for the Button component within the container:

Property	Value
props.text	Next Tank
position.grow	0
position.shrink	0
position.basis	80px

The following properties are set for the Thermometer component within the container:

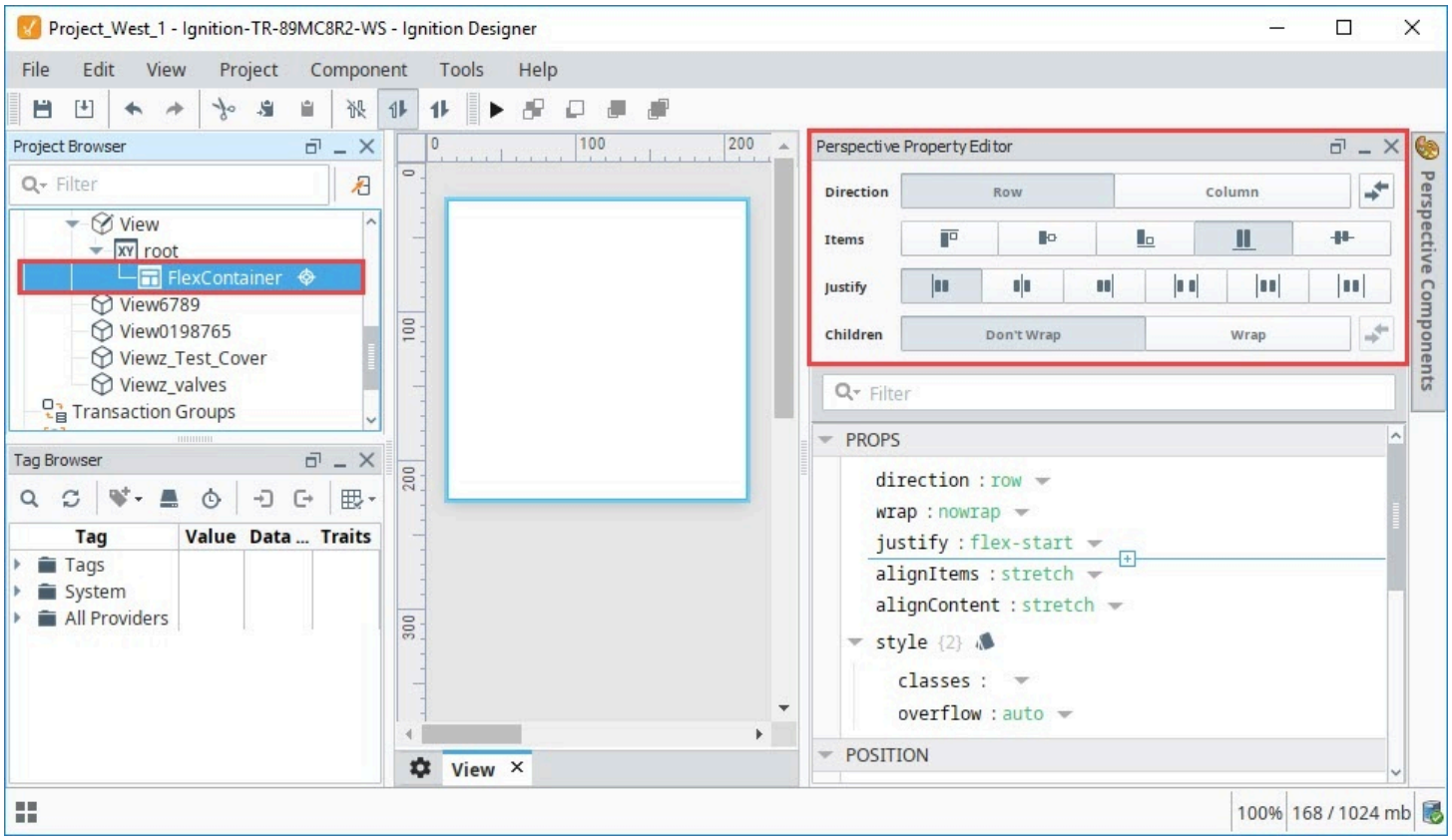
Property	Value
props.direction	row
position.grow	1
position.shrink	1
position.basis	50%

The following properties are set for the LED Display component within the container:

Property	Value
position.grow	0
position.shrink	0
position.basis	80px

Flex Container User Interface

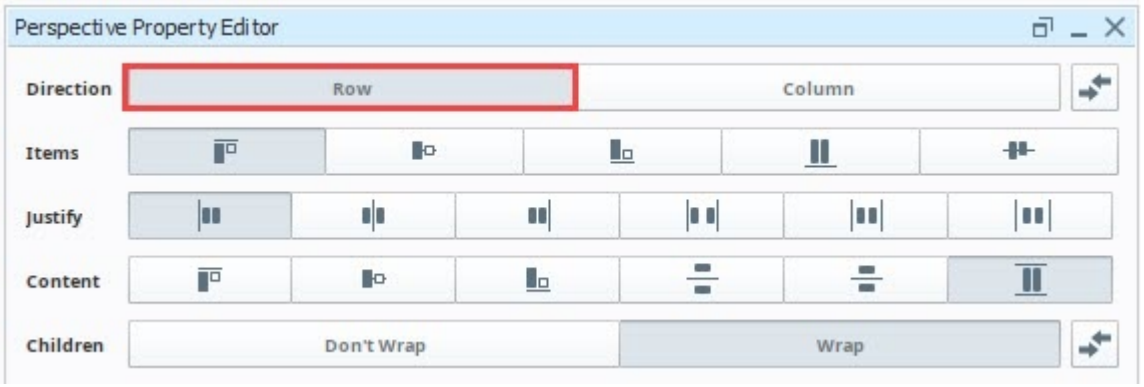
When a **Flex container** is deep selected, there is a Graphical User Interface (GUI) at the top of the Perspective Property Editor that enables you to set the container's properties. Functionality is similar to that of the properties in the Props Tree, but you may find the visual interface easier or quicker to use.



Direction

This sets the direction for the child layout. Options are **Row** or **Column**. When the **Reverse** icon is selected, the contents of this container are displayed in reverse order.

Direction: Row



The following table shows the icons and properties they represent when **Direction: Row** is selected. The icon that's displayed if Reverse order is selected is also shown.

> **NOTE**
>
> Left/right/top/bottom notes in the descriptions refer to non-reversed directions. The phrase "when there is extra space" means when no components have are stretching to fill the space. That is, when no components have "grow" greater than 0.

Items | **Row Icon** | **Row Reversed Icon** | **Property** | **Description** | | :--: | :--:| -------- | ---------- | | | | Flex Start | Child items are placed along the start (top) of the container when there is extra space. | | | | Center | Child items are placed along the center of the container when there is extra space. | | | | Flex End | Child items are placed along the end (bottom) of the container when there is extra space. | | | | Stretch | Child items are stretched from top to bottom of the container. | | | | Baseline | Child items are placed so the baseline of the text matches for all of them when there is extra space.|

Justify

Row Icon	Row Reversed Icon	Property	Description
		Flex start	Adjusts placement of children to the start (left) of the container when there is extra space. If reversed, children are placed along the right.
		Center	Adjusts placement of children along the center of the container when there is extra space.
		Flex End	Adjusts placement of children along the end (right) of the container when there is extra space. If reversed, children are placed along the left.
		Space Between	Adjusts placement of children with space in between them reaching to the edges of the container when there is extra space.
		Space Around	Adjusts placement of children with even spacing in between them with some space along the edges when there is extra space.
		Space Evenly	Adjusts placement of children with even spacing in between them and the edges of the container when there is extra space.

Children

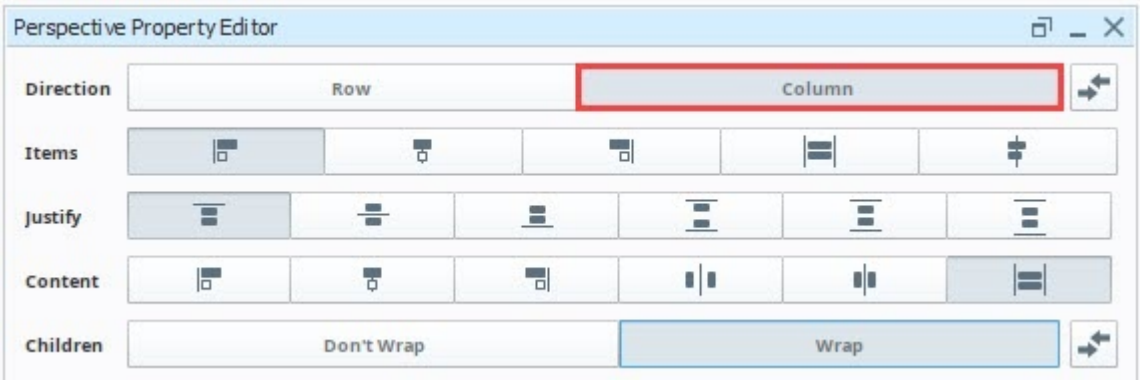
Icon	Property	Description
N/A	Don't Wrap	If there are more components than the width allows, shrink them.
N/A	Wrap	If there are more components than the width allows, wrap onto the next line.
	Reverse Wrap	Toggle to reverse the direction of wrap from top-down to bottom-up

Content

The following properties are only applicable when children: **Wrap** is selected.

Row Icon	Row Reversed Icon	Property	Description
		Flex start	Adjusts placement of wrapped content to the start (top) of the container when there is free space.
		Center	Adjusts placement of wrapped content to the middle of the container when there is free space.
		Flex End	Adjusts placement of wrapped content to the end (bottom) of the container when there is free space.
		Space Between	Adjusts placement of wrapped content evenly with space in between each wrapped line, reaching to the edges (top and bottom) of the container when there is extra space.
		Space Around	Adjusts placement of wrapped content evenly with space in between each wrapped line and the edges (top and bottom) of the container when there is extra space.
		Stretch	Adjusts placement of wrapped content evenly with space in between each wrapped line and after the last line (bottom) of the container when there is extra space.

Direction: Column



The following table shows the icons and properties they represent when **Direction: Column** is selected. The icon that's displayed if Reverse order is selected is also shown.

Items

Column Icon	Column Reversed Icon	Property	Description
		Flex start	Child items are placed along the start (left) of the container when there is extra space.
		Center	Child items are placed along the center of the container when there is extra space.
		Flex End	Child items are placed along the end (right) of the container when there is extra space.
		Stretch	Child items are stretched from left to right of the container.
		Baseline	Child items are placed so the baseline of the text matches for all of them when there is extra space.

Justify

Column Icon	Column Reversed Icon	Property	Description
		Flex start	Adjusts placement of children to the start (top) of the container when there is extra space. If reversed, children are placed along the bottom.
		Center	Adjusts placement of children along the center of the container when there is extra space.
		Flex End	Adjusts placement of children along the end (bottom) of the container when there is extra space. If reversed, children are placed along the top.
		Space Between	Adjusts placement of children with space in between them reaching to the edges of the container when there is extra space.
		Space Around	Adjusts placement of children with even spacing in between them with some space along the edges when there is extra space.
		Space Evenly	Adjusts placement of children with even spacing in between them and the edges of the container when there is extra space.

Children

Icon	Property	Description
N/A	Don't Wrap	If there are more components than the width allows, shrink them.
N/A	Wrap	If there are more components than the width allows, wrap onto the next line.
	Reverse	Toggle to reverse the direction of wrap from top-down to bottom-up

Content The following properties are only applicable when children: **Wrap** is selected.

Column Icon	Column Reversed Icon	Property	Description
		Flex start	Adjusts placement of wrapped content to the start (left) of the container when there is free space.
		Center	Adjusts placement of wrapped content to the middle of the container when there is free space.
		Flex End	Adjusts placement of wrapped content to the end (bottom) of the container when there is free space.
		Space Between	Adjusts placement of wrapped content evenly with space in between each wrapped line, reaching to the edges (left and right) of the container when there is extra space.
		Space Around	Adjusts placement of wrapped content evenly with space in between each wrapped line and the edges (left and right) of the container when there is extra space.
		Stretch	Adjusts placement of wrapped content evenly with space in between each wrapped line and after the last line of the container when there is extra space.

Perspective - Flex Container Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Flex Container** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Functions

.getChildren()

- Description
  - Returns an ArrayList, which contains references to all components inside of the container.
- Parameters
  - None
- Return
  - Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

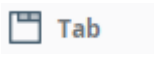
Extension Functions

This component does not have extension functions associated with it.

Perspective - Tab Container



Component Palette Icon:



Description

The Tab container uses tabs as navigation buttons arranged together with the selected tab highlighted. Only one component can be displayed in each tab.

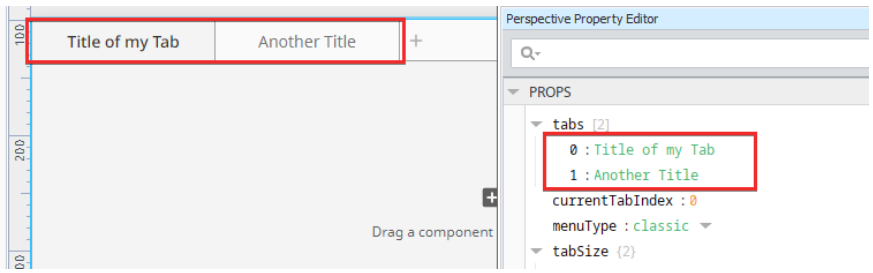
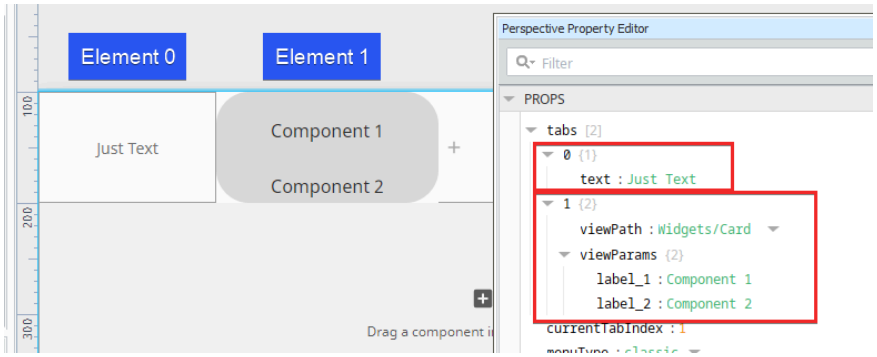
The Tab Container component has two pre-configured **variants**:

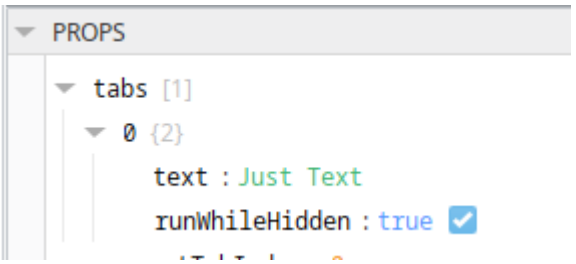
- Classic - Layout is a traditional menu with boxed tabs.
- Modern - Layout has no borders around each tab and shows selection with an underline.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are

described on the **Perspective Component Properties** page.

Property	Description	Property Type
tabs	Responsible for the number of tabs in the container. Adding additional elements to this array will result in an additional tab being rendered on the component. In addition, each element is responsible for determining the content of the tab: either text or an embedded view. By default, the component starts with a single element is this array, set to a value type. Providing a string to the element will set the text on the tab.  If the element is changed to an object data type, then the text on the tab can be set by adding a **text** value member to the object and providing a string, as seen on element 0 below. Alternatively, **viewPath** (value data type) and **viewParams** (object data type) can be added to the element, which allows you to render a view inside the tab as well as pass parameters to it. Element 1 below demonstrates the idea. Keep in mind that the viewPath property will be ignored in favor of the text property if both exist in the object.  A boolean **runWhileHidden** property can be added to an element that's set to an object data type. This setting determines if contents of this tab loads once the tab is first	array

Property	Description	Property Type
	activated, and if it should persist while in the background when the **currentTabIndex** changes.  A boolean **disabled** property can be added to an element that's set to an object data type. This setting determines if a tab can be selected. If the tab is already active when it becomes disabled, users will not be prevented from leaving the tab, but they will not be able to get back to it. The content of the tab is not affected by the enabled/disabled state of the tab.	
currentTabIndex	Which index in tabs array is currently active.	value: numeric
menuType	If the type is 'classic', a traditional menu with boxed tabs is shown. If the type is 'modern', it has no borders around each tab and shows selection with an underline.	value: string dropdown
tabSize	Default size allotted to a single tab. If a container width does not allow, tab width will shrink from this size accordingly. • **width**: Width in pixels for the tab size, numeric property type. • **height**: Height in pixels for the tab size, numeric property type.	object
menuStyle	Opens the Style menu to change Tab properties: Text, Background, Margin and Padding, Border, and Misc.	value: string dropdown
contentStyle	Sets a style for the content frame component. A full menu of **style options** are available. You can also specify a **style**	object

Property	Description	Property Type
	class.	
tabStyle	Additional styling to apply to all tabs depending active (selected) or inactive state. • **active**: Sets a style for a tab when it is the active tab. The Style menu contains all the tools for modifying text, background, margins, borders, and more. You can also specify a style class, object property type. • **inactive**: Sets a style for tabs that are inactive. The Style menu contains all the tools for modifying text, background, margins, borders, and more. You can also specify a style class, object property type.	object
style	Use Style to customize the visual style of the component. A full menu of style options are available. You can also specify a style class.	object

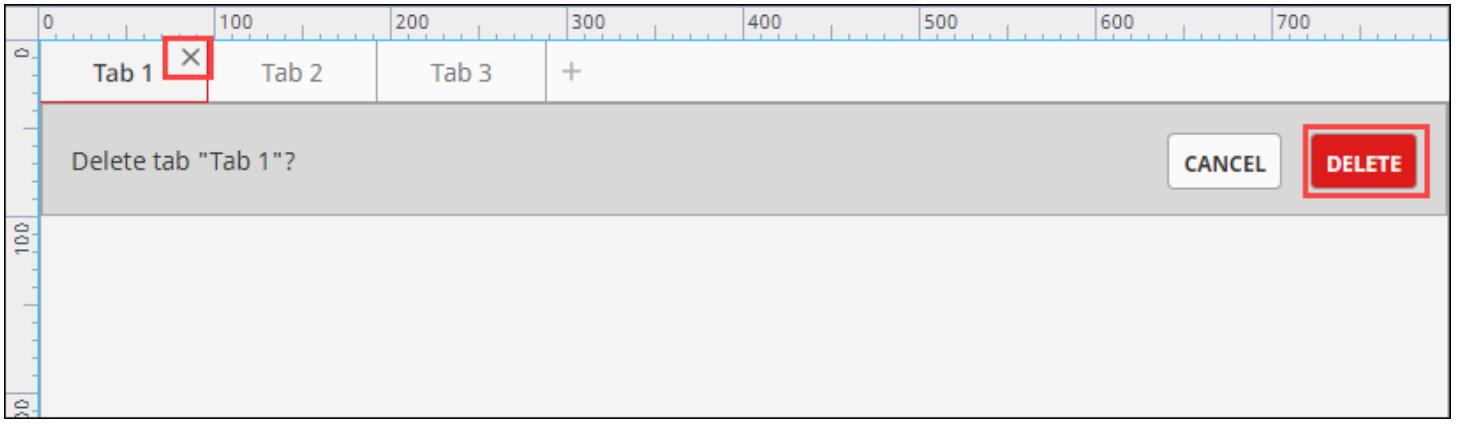
User Interaction

Adding Components to Tabs

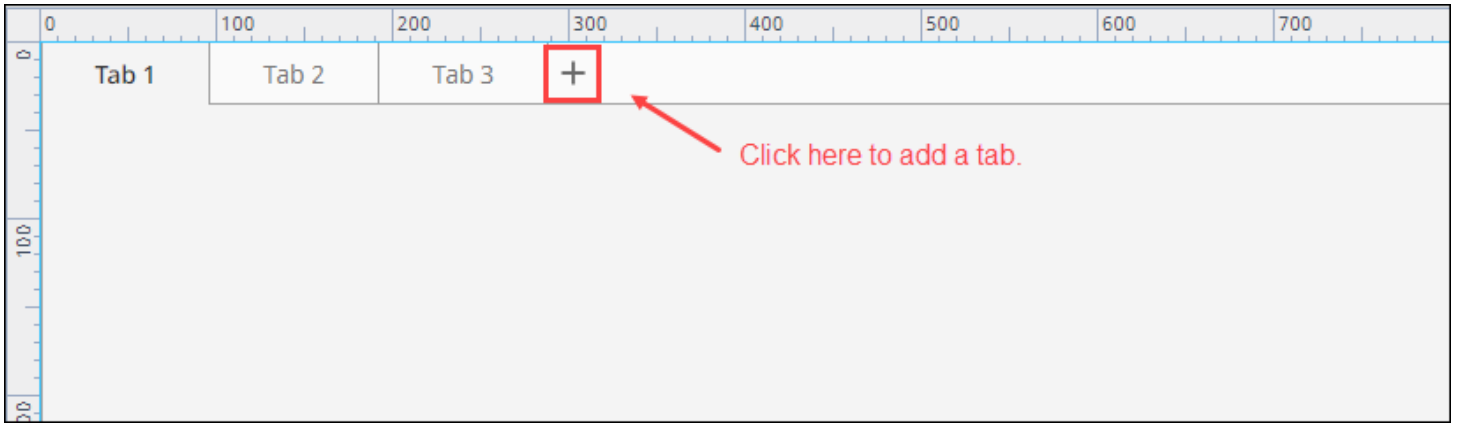
To add a component to a tab, deep select the Tab container, then drag a component onto it.

Adding and Deleting Tabs

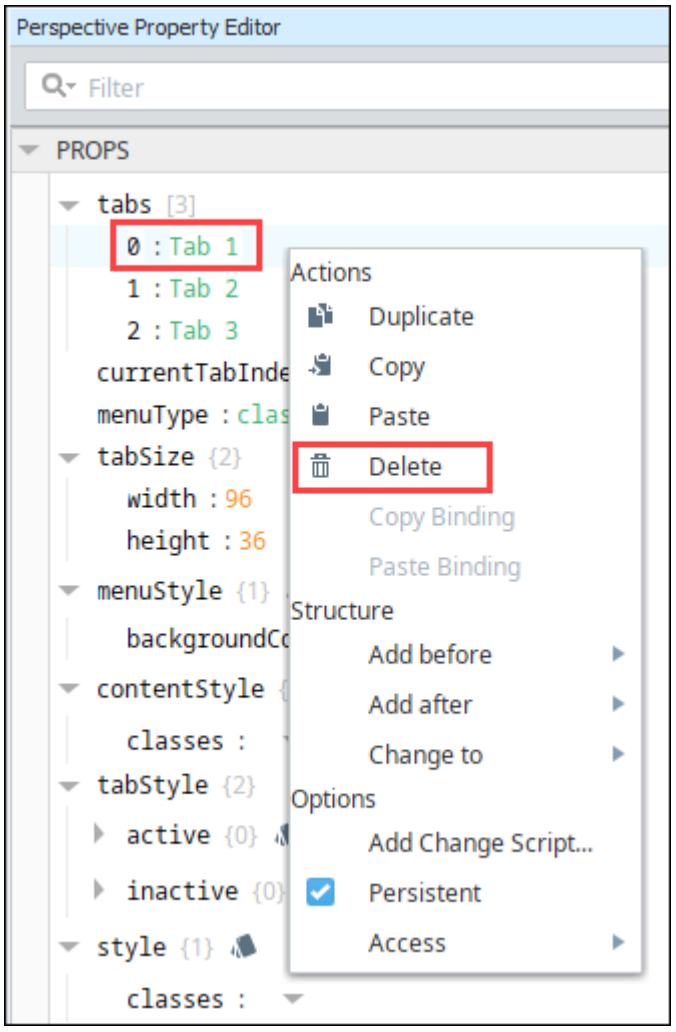
To delete a tab, click on the "X" to the right of the tab itself.



To add a tab, click on the Add icon to the right of the tabs:



You can also use the right-click menu in the Property Editor. Just right click on the tab you want to work with. You'll see options for copying, pasting,

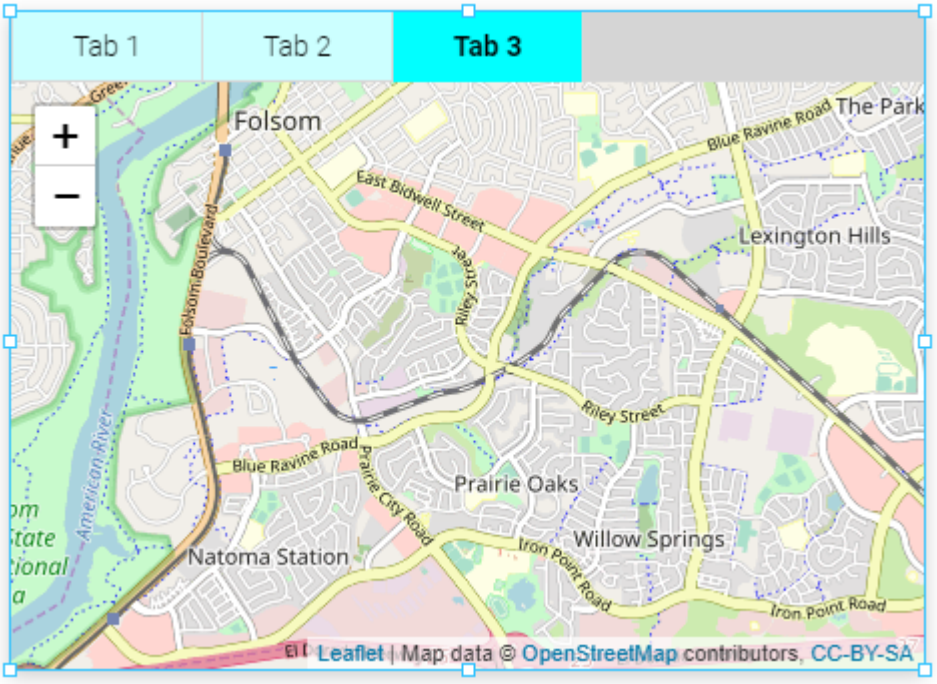


Scripting

See the **Perspective - Tab Container Scripting** page for the full list of scripting functions available for this component.

Examples

Example 1

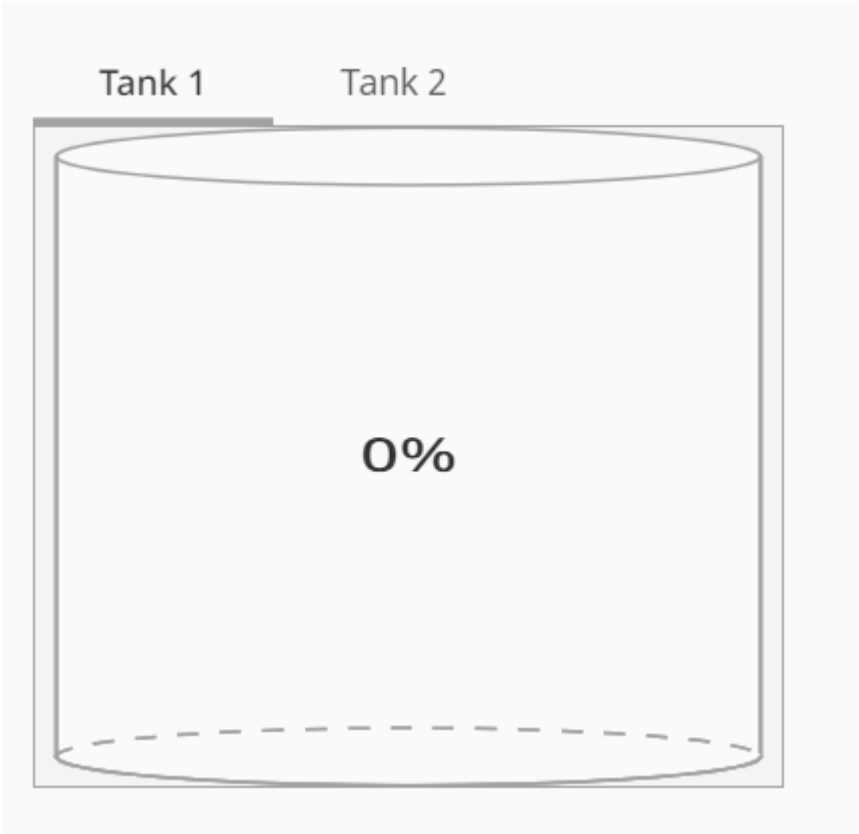


In this example, the default (Classic variant) Tab Container is used. Three tabs are set up in the Tab Container component. Tab 3 is active and contains a Map component.

Property	Value
props.tabs.0	Tab 1
props.tabs.1	Tab 2
props.tabs.2	Tab 3
props.menuStyle.backgroundColor	#D5D5D5
tabStyle.active.backgroundColor	#00FFFF
tabStyle.active.fontWeight	bold
tabStyle.inactive.backgroundColor	#CCFFFF
tabStyle.inactive.fontWeight	lighter

Example 2

In this example, the default (Modern variant) Tab Container is used. Two tabs are set up in the Tab Container component, each with a Cylindrical Tank component.



Property	Value
props.tabs.0	Tank 1
props.tabs.1	Tank 2
props.menuType	modern

Perspective - Tab Container Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Tab Container** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Functions

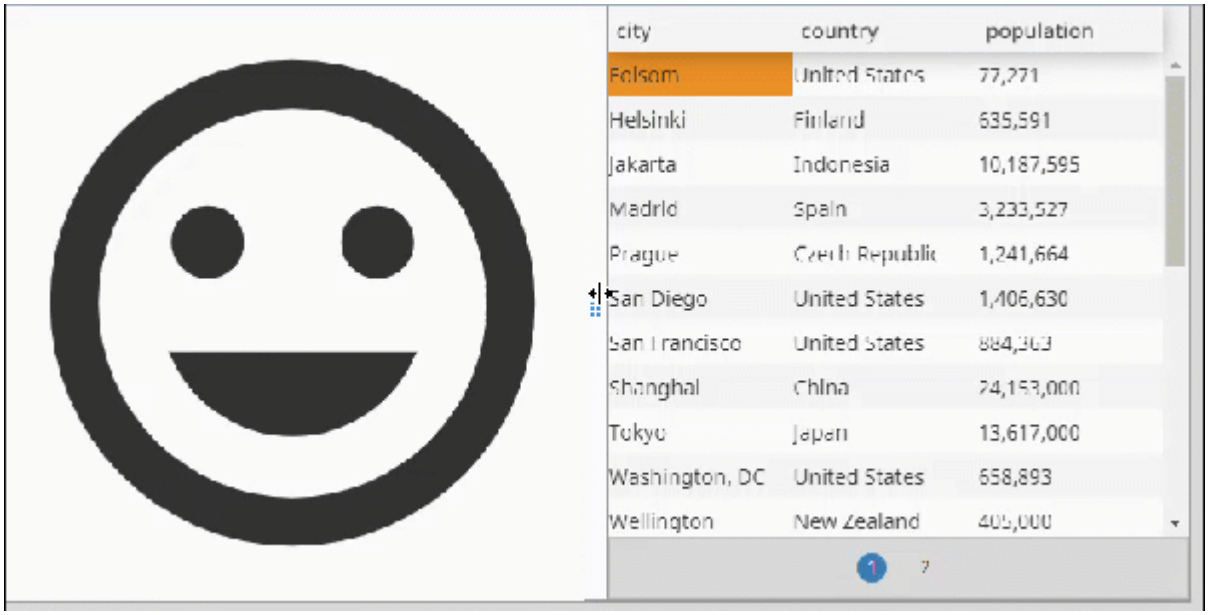
.getChildren()

- Description
  - Returns an ArrayList, which contains references to all components inside of the container.
- Parameters
  - None
- Return
  - Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

Extension Functions

This component does not have extension functions associated with it.

Perspective - Split Container



Component Palette Icon:



Description

The Split container holds two children separated by a draggable "split" that allows the user to resize the two children during the runtime. The Split container supports both horizontal and vertical layouts.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Property	Description	Property Type
orientation	The orientation in which the container's split is fixed. Horizontal will allow the user to adjust the container from left to right, while Vertical will allow the user to adjust it from top to bottom.	value: numeric
split	Configuration for the split bar. • **position**: Bidirectionally represents the position of the split bar. Numeric values here will be used as pixels, numeric or string property type. • **size**: The size of the split bar in pixels, numeric property type. • **visible**: Determines if the split bar should be visible, boolean property type. • **draggable**: Determines if the split bar should be draggable, boolean property type.	object
style	Use Style to customize the visual style of the component. The **Style menu** contains all the tools for modifying text, background, margins, and borders. You can also specify a **style class**.	object

Child Component Position Properties

When a component is placed inside of a Split container, it will inherit the position property listed below.

Property	Description	Data Type
position	Indicates which side of the split bar the child component is located on. Expected values include "left", "right", "top", and "bottom"	value: string

Scripting

See the **Perspective - Split Container Scripting** page for the full list of scripting functions available for this component.

Perspective - Split Container Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Split Container** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onMinBoundReached

This event is fired when the split reaches the minimum bound on the container. For example, when the split bar's position reaches 0 px or 0% of the container's width.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

onMaxBoundReached

This event is fired when the split reaches the maximum bound on the container. For example, when the split bar's position reaches 100% of the container's width.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

Component Functions

.getChildren()

- Description
  - Returns an ArrayList, which contains references to all components inside of the container.
- Parameters
  - None
- Return
  - Array List - An ArrayList of components in the container. The resulting ArrayList can be iterated over via a for-loop.

Extension Functions

This component does not have extension functions associated with it.

Perspective - Display Palette

Perspective offers a variety of components that display static and dynamic information.

Here is a complete list of Display components, and a link pointing to a page containing the component's description, properties, and an example of how to configure it.

Alarm Journal Table

Alarm Status Table

Audio

Barcode

The Barcode component enables you to display text as a barcode.

Cylindrical Tank

A component that looks like a 3D cylindrical tank with some liquid inside.

Dashboard

Drawing

Equipment Schedule

Google Map

Icon

The Icon component provides access to a collection of Scalable Vector Graphic (SVG) images.

Image

The Image component displays either vector or raster format images, such as a jpeg, gif, png, or svg.

Inline Frame

The Inline Frame component enables you to display a webpage within the component.

Label

The Label component displays text and can be customized.

LED Display

The LED Display is a stylized numeric and/or alphanumeric label.

Linear Scale

The Linear Scale component displays a series of tick marks and labels.

Map

Markdown

The Markdown component allows users to format any type of text so it is publishable as plain text.

Moving Analog Indicator

The Moving Analog Indicator displays an analog value in context with other information.

PDF Viewer

Progress

The Progress bar visually indicates the progress of a task.

Sparkline

The Sparkline chart is a minimalistic chart component that displays a line-chart history for a single datapoint.

Table

Tag Browse Tree

Thermometer

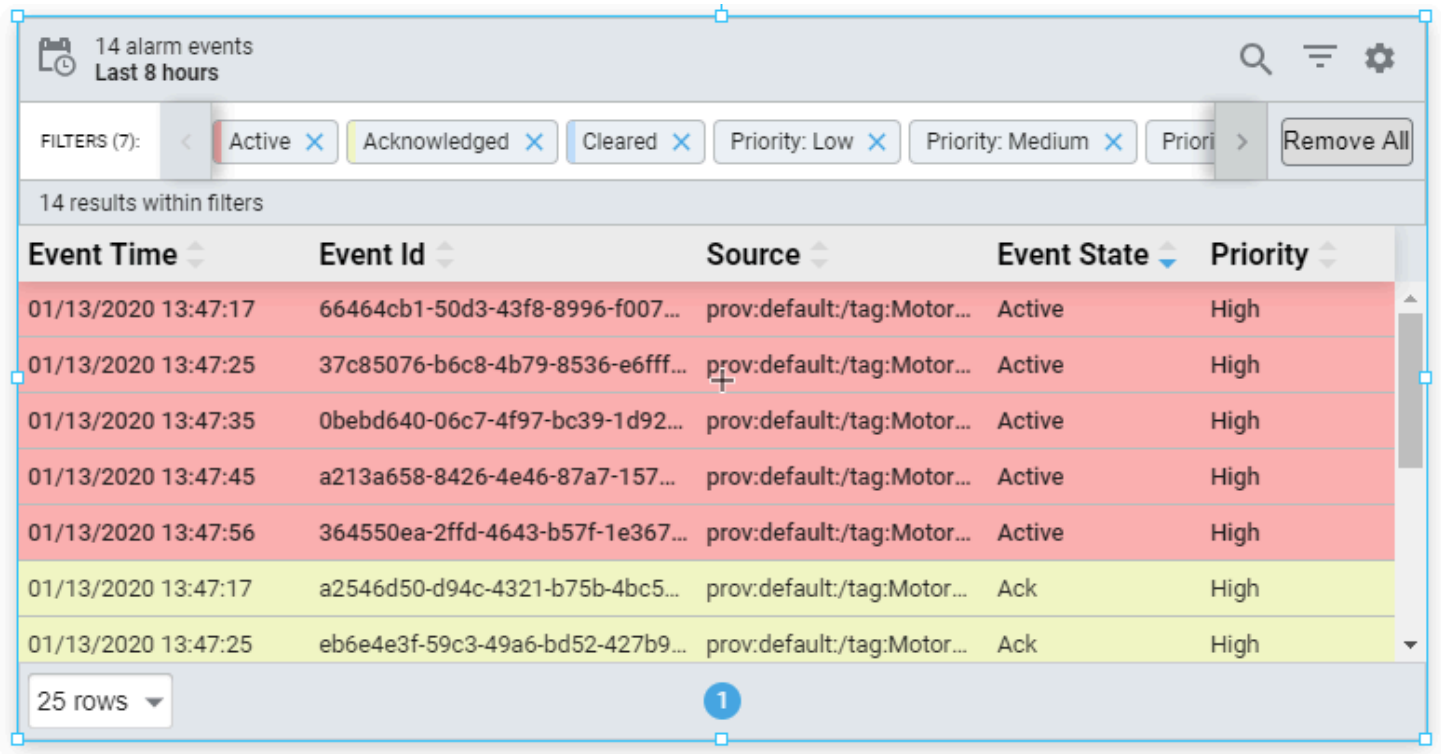
The Thermometer component displays a temperature value depicted as a level in a mercury thermometer.

Tree

Video Player

The Video Player component enables you to embed video or a live feed in Perspective views.

Perspective - Alarm Journal Table



Component Palette Icon:



Description

The Perspective Alarm Journal Table displays the history of the alarm system. It can be configured to show active, cleared, and acknowledged events.

Before the Alarm Journal Table can retrieve alarm data, an **Alarm Journal** must first be configured.

The Perspective Alarm Journal Table has a number of configuration options that can be used to filter on realtime and historical alarm data, and change how the component displays those alarms. When you first drag the Alarm Journal Table into the Designer, by default, the table will show you the last 8 hours of alarm journal data. You can interface with the journal table in the Designer, in Preview Mode of the Designer, and in a Perspective Session.

The Alarm Journal Table provides a host of filtering properties that allow you to filter on various parts of alarms and view the details. All the alarm states are visible by clicking the Filter button on the table.

There is also a search bar where you can enter text to further refine your filter criteria so you have less alarm events to scroll through. The Alarm Journal Table can filter for alarm events in either Realtime or Historical using the Date Range feature.

You can change the columns that are displayed, the order of the columns, and/or the column width in Preview Mode and in a Perspective Session. Right-click on the table header to show/hide columns. Click and drag the margins of the columns to resize their width. You can also sort table columns in ascending or descending order by simply clicking the up or down arrows next to each column header.

The Alarm Journal Table utilizes a shared polling engine when in realtime date range mode to cache and share polling tasks across concurrent sessions. The cache persists for a period of time that matches the configured poll rate. This optimization allows multiple components to poll the alarm system with a reduced impact on overall performance.

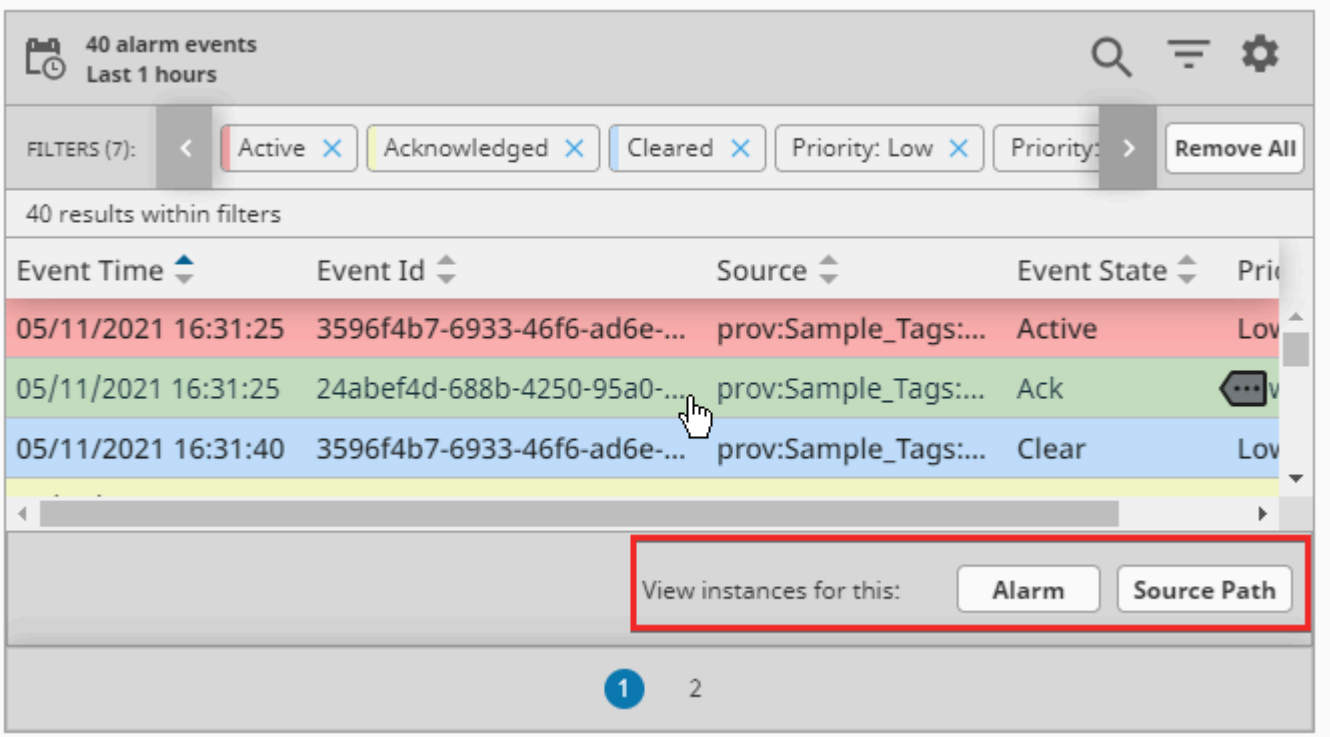
User Interface

The following table describes the user interface available for the Alarm Journal Table.

Interaction	Description
Config Settings	Clicking the  icon or right clicking on a column heading will open the Configuration menu, which determines which columns show appear on the component.
Date Range	Clicking on the  icon will open the Date Range window, allowing the user to determine a range of time to filter on. Alarm events with an Event Time that matches the selected range will appear on the table.
Filters	Clicking the  icon opens the Filters menu, allowing you to select event states and priorities to filter on.
Pages	A listing of pages. Long lists of alarm events are spread across multiple pages to improve performance. Clicking on a number will switch which page is shown.
Popup Modal	Hovering over a row in the table will cause the Popup Modal icon to appear. Clicking this icon shows more information about the alarm.
Rows to Display	Determines how many rows are shown per page.
Search	Clicking the  icon will cause a search bar to appear, allowing the user to type in search terms.

Additional interface options become available when selecting a row within the table, which allow the table to search for similar events.

- Alarm: Shows all events for a single alarm instance, meaning the active event, the clearing event, and the acknowledging event (if present).
- Source Path: Shows all events that match the same source path as the selected event, respecting the selected date range.



Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
name	The exact name of the **alarm journal** to query that was set up in the Gateway. Default is "Journal". **Note:** In Gateway Network configurations, set this to an empty string OR use the name of the Remote Alarm Journal. On Edge Gateways, set this to an empty string value to have the component use the Edge Alarm Journal.	value: string
refreshRate	The rate at which the table will poll for updates in milliseconds.	value: numeric
enableHeader	Enables the table header. Default is true.	value: boolean
enableDetails	Enables the details action. Default is true.	value: boolean
orderByAscending	Sorts alarms in ascending order by the event time when true. Default is true.	value: boolean
toolbar	Settings for the toolbar. Click to see the **toolbar** properties	object
dateFormat	A date format string to be applied against dates. Options are none, date in the format "10/15/2018", time in format "3:59:00 PM", or date time in format "10/15/208 15:59:00".	value: string
responsive	Responsive layout configuration. Rows are converted to cards. While in responsive layout, disables or removes certain table features that are no longer applicable. Click to see the **responsive** properties	object
dateRange	Settings for date range state. Click to see the **dateRange** properties	object
filter	Filter settings. Click to see the **filter** properties	object

Name	Description	Property Type
rowStyles	Styles to apply to rows given their alarm event and designated priority. Click to see **rowStyles** properties	object
dragOrderable	When enabled, users may drag column headers to reorder columns in the table.	value: boolean
columns	Used for determining what column properties to display on the Alarm Journal Table. Click to see **columns** properties	object
columnsAssociated	A list of columns used to retrieve and display alarm associated data. Click to see **columnsAssociated** properties.	object
sortOrder	The default weighted order in which columns and their contents are sorted relative to other columns and their contents. Only works if used when the component loads. Columns need to have sort configured in order for this to work.	array
pager	Settings for the pager. Click to see **pager** properties.	object
style	Sets a style that applies to the component. The **Style menu** contains all the tools for modifying text, background, margins, and borders. You can also specify a **style class**.	object

toolbar Properties

Name	Description	Property Type
enabled	Enables the visibility of the table toolbar. Default is true.	value: boolean
enableDateRange	Enables the visibility of the date range toggle. Default is true.	value: boolean
enableFilter	Enables the visibility of the text filter toggle. Default is true.	value: boolean
toggleableFilter	If false, the text filter will not require user interaction to open, and instead will remain open. Default is true.	value: boolean
enableFilterResults	Enables the visibility of the filters results count message. Default is true.	value: boolean
enablePreFilters	Enables the visibility of the prefilter toggle. Default is true.	value: boolean
enableConfiguration	Enables the visibility of the configuration toggle. Default is true.	value: boolean

responsive Properties

Name	Description	Property Type
enabled	Enables responsive layout. Default is false.	value: boolean
breakpoint	Width in pixels that triggers change in responsive layout.	value: numeric

dateRange Properties

Name	Description	Property Type
mode	The current date range mode: realtime or historical.	value: string
realtime	When mode is set to **realtime**, this property will appear with the following sub properties available: • **interval**: The realtime interval as an integer, value is numeric. • **unit**: The realtime interval unit: seconds, minutes, hours, days, months and years, value is a string.	object
historical	When mode is set to **historical**, this property will appear with the following sub properties available: • **startDate**: The start date to use, in milliseconds. Value is numeric • **endDate**: The end date to use, in milliseconds. Value is numeric.	object

filter Properties

Name	Description	Property Type
text	The alarm events filter text.	value: string
events	Alarm event types. All of the following properties are boolean values. • **active**: Whether to display alarm events with active states. Default is true. • **acked**: Whether to display alarm events with acked states. Default is true. • **cleared**: Whether to display alarm events with cleared states. Default is true. • **enabled**: Whether to display alarm events that were generated by enabling an alarm. • **disabled**: Whether to display alarm events that were generated by disabling an alarm. • **system**: Whether to display alarm event with system states. Default is false.	object
priorities	Alarm event priority pre-filters. All of the following properties are boolean values. • **diagnostic**: Whether to display alarms with diagnostic priority. Default is false. • **low**: Whether to display alarms with low priority. Default is true. • **medium**: Whether to display alarms with medium priority. Default is true. • **high**: Whether to display alarms with high priority. Default is true. • **critical**: Whether to display alarms with critical priority. Default is true.	object
conditions	Gateway side alarm query conditions. All of the following properties are string values.	object

Name	Description	Property Type
	• **source**: Filter alarms by alarm source path. Specify multiple paths by separating them with commas. Supports the wildcard *. • **displayPath**: Filter alarms by display path, falling back to the source path if a custom display path in not set. Specify multiple paths by separating them with commas. Supports the wildcard *. • **provider**: Filter alarms by provider. Refer to the **Filtering Alarms** section on the Alarming page for more information on displayPath and source syntax.	
results	Alarm event filtering results configuration and data. • **enabled**: Enable filtering results to be written back to props. Value is a boolean. **Note**: Enabling this property may cause a performance decline. • **data**: An array of objects representing the current filtered data, if enabled and active, array property type.	object

rowStyles Properties

Name	Description	Property Type
active	Style settings for rows with active alarms.	object
acked	Style settings for rows with acked alarms.	object
cleared	Style settings for rows with cleared alarms.	object
system	Style settings for rows with system alarms.	object

rowStyles Child Properties

Each rowStyles property has the following child properties. For each, a full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**

Name	Description	Property Type
base	Base style settings for alarms.	object
priorities	Style settings for the alarm row based on priority. • **diagnostic**: Style for acked alarms with diagnostic priority. • **low**: Style for acked alarms with low priority. • **medium**: Style for acked alarms with medium priority. • **high**: Style for acked alarms with high priority. • **critical**: Style for acked alarms with critical priority.	object

columns Properties

Name	Description	Property Type
eventTime	Settings for the eventTime column.	object
eventId	Settings for the eventId column.	object
name	Settings for the name column.	object
source	Settings for the source column.	object
eventState	Settings for the eventState column.	object
priority	Settings for the priority column.	object
state	Settings for the state column.	object
displayPath	Settings for the displayPath column.	object
label	Settings for the label column.	object
eventValue	Settings for the eventValue column.	object
isSystemEvent	Settings for the isSystemEvent column.	object
ackUser	Settings for the ackUser column.	object
ackNotes	Settings for the ackNotes column.	object

column Child Properties

Each column type property has the following child properties.

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

columnsAssociated Properties

Name	Description	Property Type
field	Maps to the associated data value represented by the column.	value: string
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string
order	Order to display this column in the table.	value: numeric

pager Properties

Name	Description	Property Type
enabled	Enables the pager to be displayed. Default is true.	value: boolean
hide	Visually hides the pager from view. Useful when pager is manipulated in a controlled fashion via the activePage property. Default is false.	value: boolean
options	Rows to show per pager option.	array
activeOption	The option to use for the table. It must exist as an available option. If the table changes as a result of user interaction, the table will write to this property internally and reflect the user's selections. External writes to this property are also possible, and will update the table accordingly. **Changed in 8.3.10** The `activeOption` property deprecates the `initialOption` property.	value: numeric
activePage	Represents the current active page and corresponds to the value of the page jump input field.	value: numeric

Scripting

See the **Perspective - Alarm Journal Table Scripting** page for the full list of scripting functions available for this component.

Perspective - Alarm Journal Table Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Alarm Journal Table** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Functions

.refreshData()

- Description
  - Refreshes the data on the Alarm Journal Table.
- Parameters
  - None
- Return
  - Nothing

Extension Functions

.filterAlarm

- Description
  - Called for each event before it is displayed in the table, allowing you to hide or show each alarm event (row) in the table. Provides an opportunity to write a more complex filter than what's normally provided to the component. Return False to exclude an alarm event from the table.

- Parameters
  - ComponentModelScriptWrapper.SafetyWrapper **self**- A reference to the component that is invoking this function.
  - PyAlarmEvent **alarmEvent** - The alarm event itself. Call `alarmEvent.get('propertyName')` to inspect properties on the event. Common properties: `name`, `source`, `priority`.
- Return
  - Boolean - The function must return either a True or False for every alarm event in the table. True will show the alarm. False will hide the alarm.

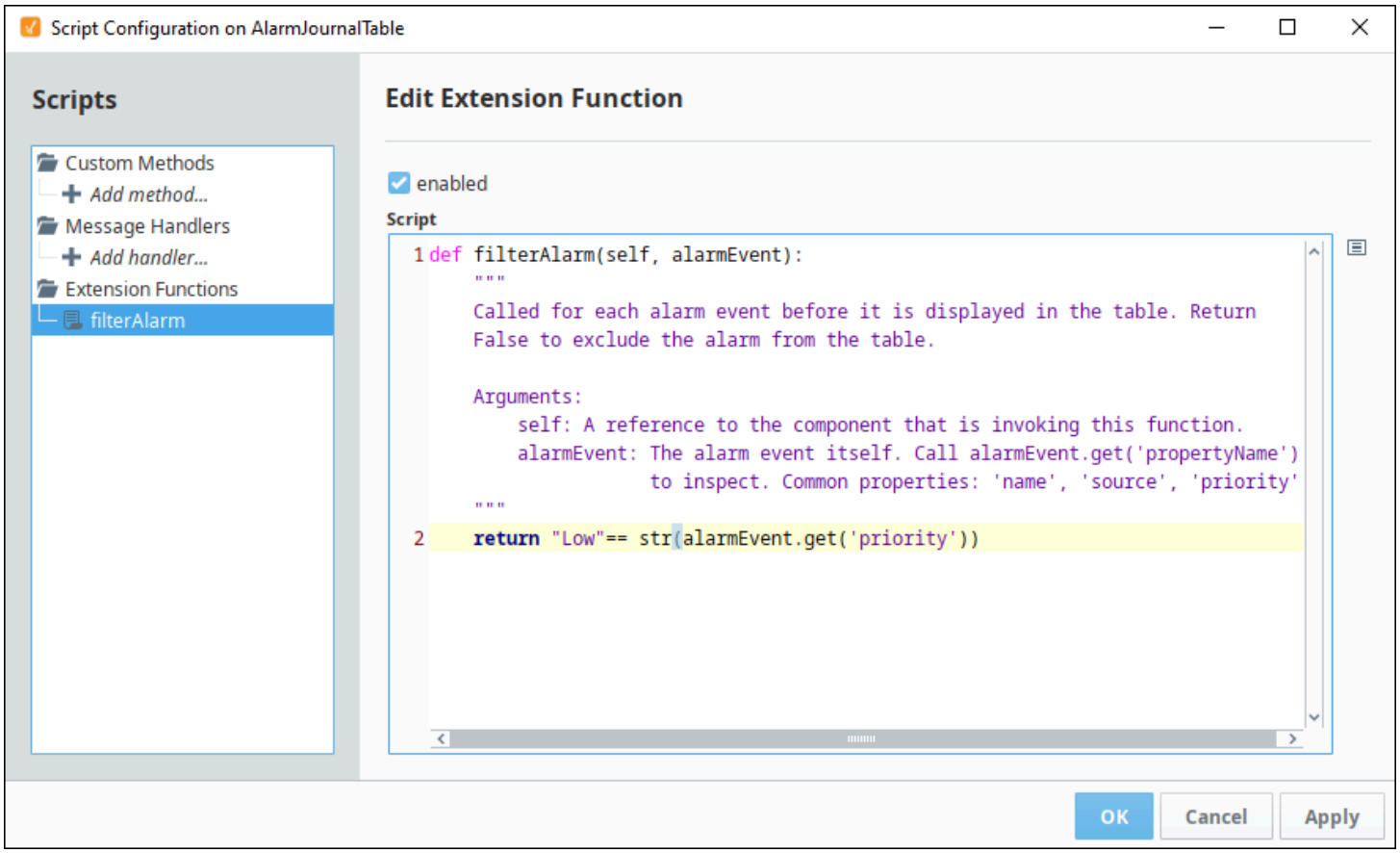
Examples

With the built-in alarmEvent object all **alarm event properties** are accessible to this function, and can be used to help determine if any given event should appear on the table. Furthermore, Associated Data (also known as custom alarm properties) can be examined from the same event.

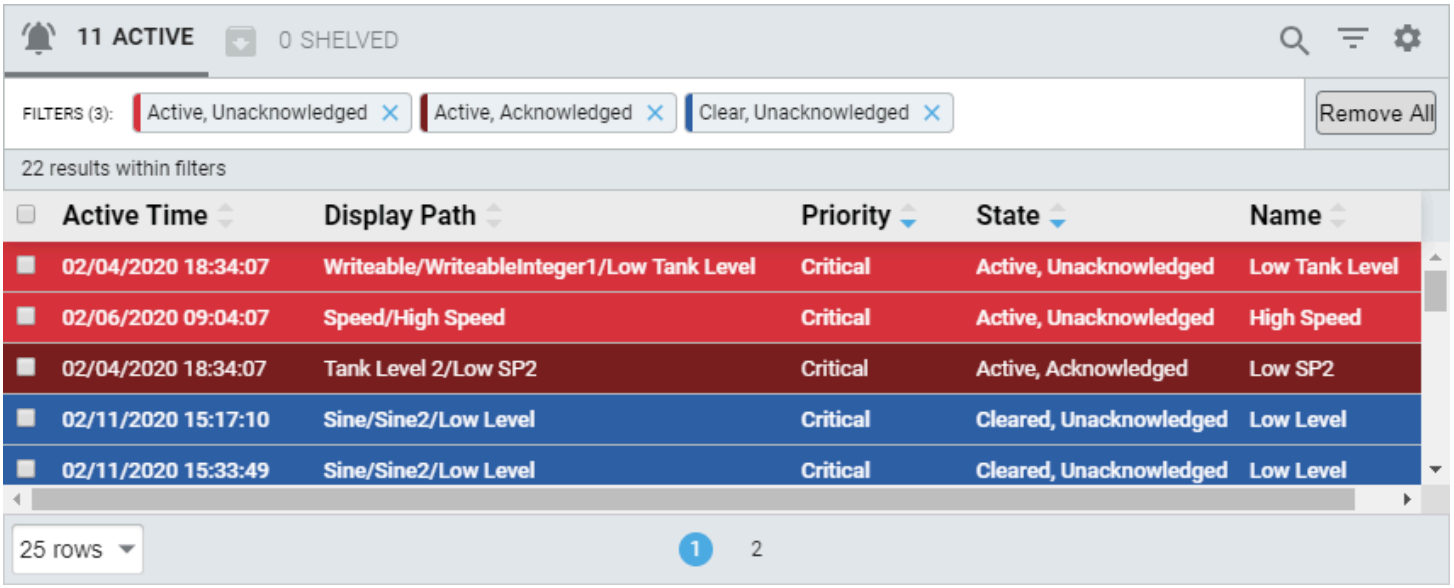
```
# Replace Property Name below with a the name of the property you wish to filter
on.
if alarmEvent.get('Property Name'):
    return True
# It's important that you return a False value for the events you don't want to
see in the table.
else:
    return False
```

You could also condense the code example above by using something like the following:

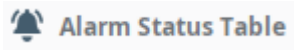
```
return "Low"== str(alarmEvent.get('priority'))
```


Script Configuration on AlarmJournalTable
Scripts
Custom Methods
Add method...
Message Handlers
Add handler...
Extension Functions
filterAlarm
Edit Extension Function
enabled
Script
1 def filterAlarm(self, alarmEvent):
"""
Called for each alarm event before it is displayed in the table. Return
False to exclude the alarm from the table.
Arguments:
self: A reference to the component that is invoking this function.
alarmEvent: The alarm event itself. Call alarmEvent.get('propertyName')
to inspect. Common properties: 'name', 'source', 'priority'
"""
2 return "Low"== str(alarmEvent.get('priority'))
OK
Cancel
Apply

Perspective - Alarm Status Table



Component Palette Icon:



Description

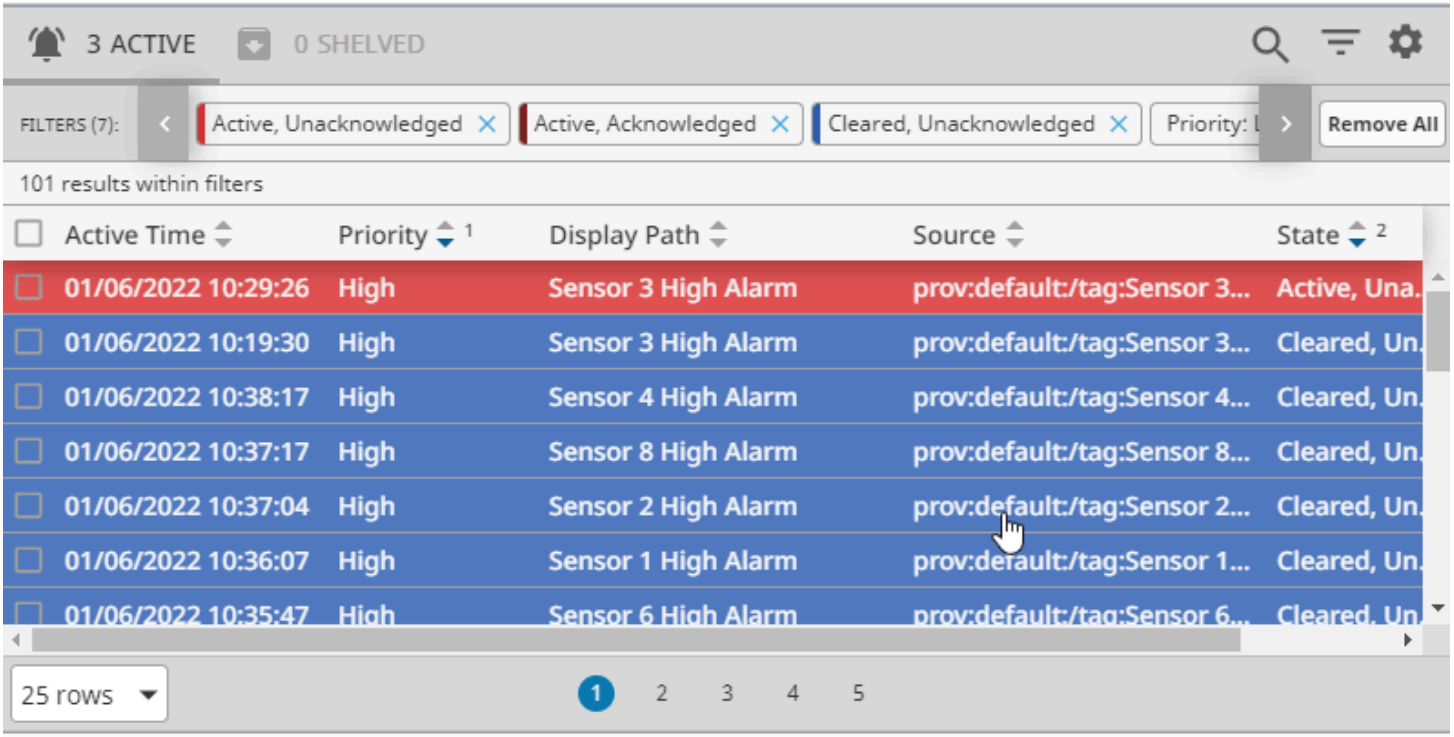
The Alarm Status Table allows you to view currently active alarm events in the system, providing an easy way to inspect the alarm details, shelve alarms, and acknowledge them.

Acknowledgement is handled by selecting (checking) alarms and pressing the **Acknowledge** button.

Shelving is supported by pressing the **Shelve** button when an alarm is selected and choosing a time duration.

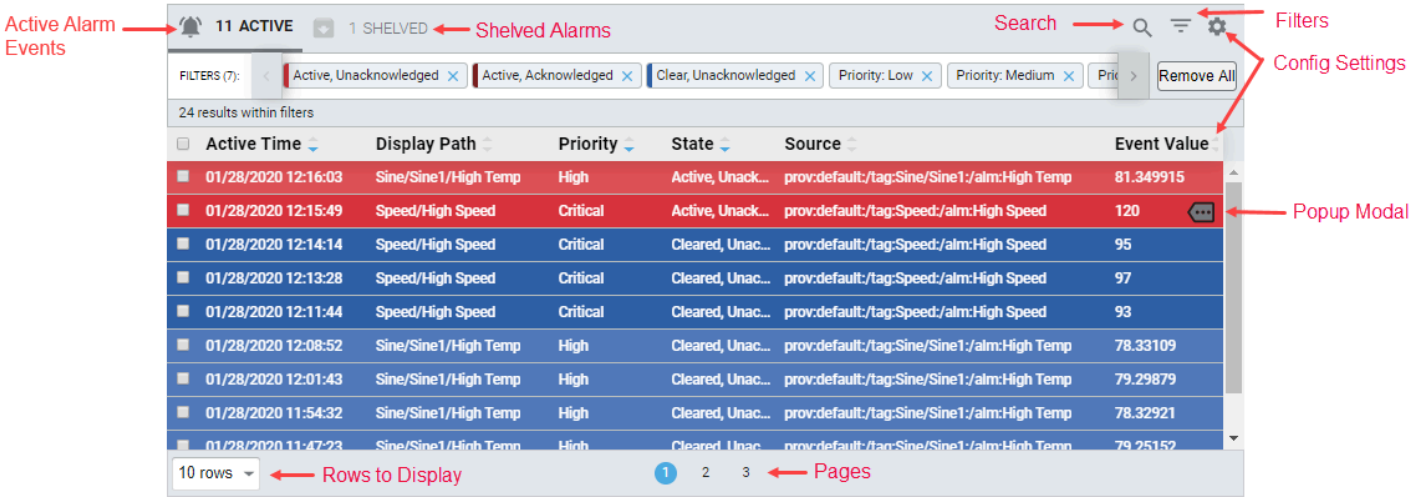
You can change the columns that are displayed and the column width in Preview Mode and in a Perspective Session. Right-click on the table header to show/hide columns. Click and drag the margins of the columns to resize their width. You can also sort table columns in ascending or descending order by simply clicking the up or down arrows next to each column header. Sorting on alarm State and Priority in the Alarm Status Table sorts in descending order. All the other columns the sort order is alphanumerical.

Columns can be reordered in Preview Mode and in a Perspective Session by clicking and dragging when the dragOrderable property is enabled.

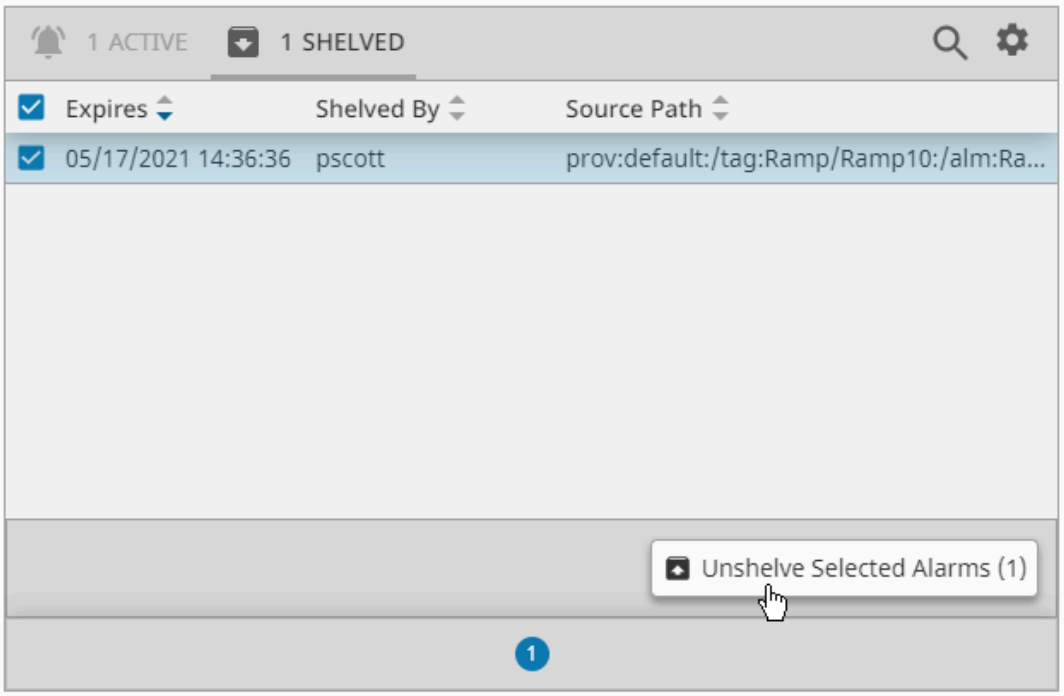


The Alarm Status Table utilizes a shared polling engine to cache and share polling tasks across concurrent sessions. The cache persists for a period of time that matches the configured poll rate. This optimization allows multiple components to poll the alarm system with a reduced impact on overall performance.

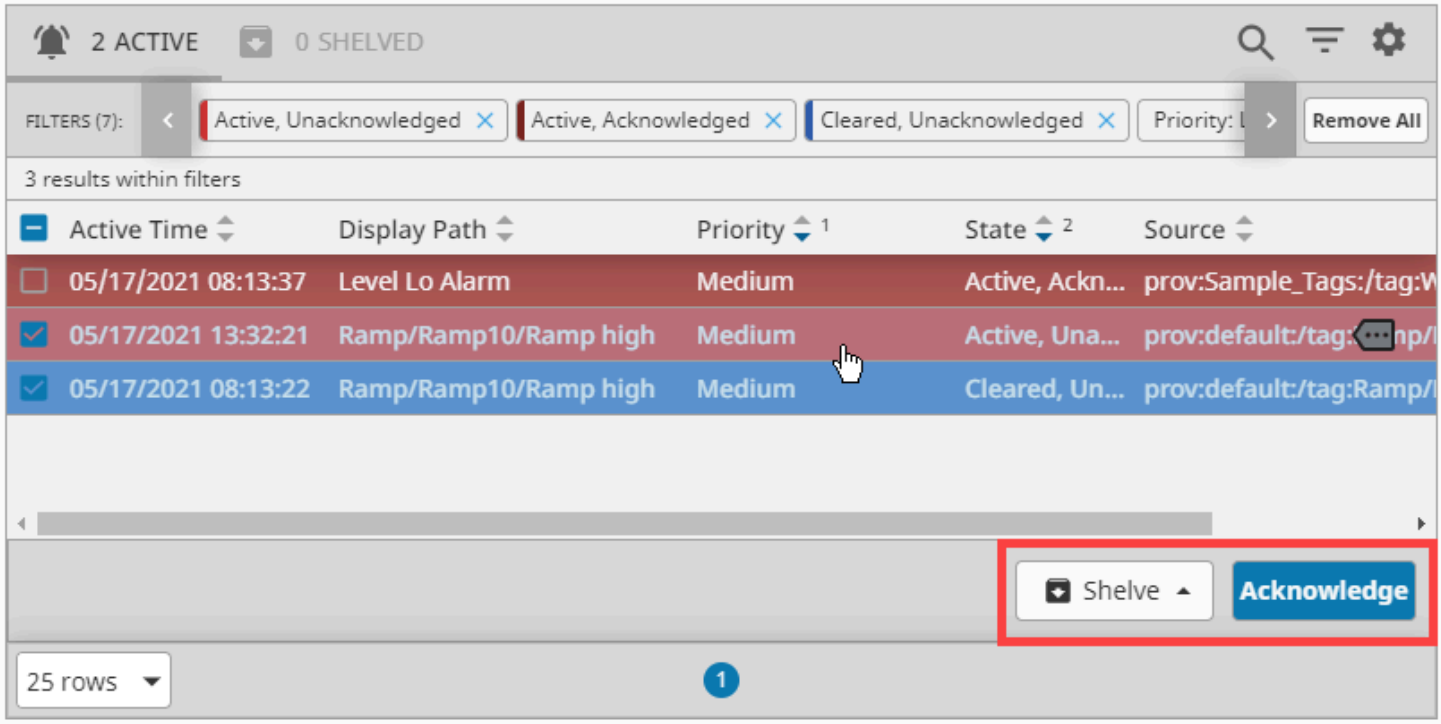
User Interface



Interation	Description
Active Alarm Events	Shows the number of active alarms in the system. When viewing the shelved alarms view, clicking on the bell icon will switch the component back to the realtime view.
Config Settings	Clicking the  icon or right clicking on a column heading will open the Configuration menu, which determines which columns show appear on the component.
Filters	Clicking the  icon opens the Filters menu, allowing you to select event states and priorities to filter on.
Pages	A listing of pages. Long lists of alarm events are spread across multiple pages to improve performance. Clicking on a number will switch which page is shown.
Popup Modal	Hovering over a row in the table will cause the Popup Modal  icon to appear. Clicking this icon shows more information about the alarm.
Rows to Display	Determines how many rows are shown per page.
Search	Clicking the  icon cause a search bar to appear, allowing the user to type in search terms.
Shelved Alarms	Shows the number of shelved alarms. When viewing the realtime view, clicking on the Shelved Alarms display will switch to the shelved alarms view. While the table is showing shelved alarms, shelved alarms can be selected from the table and unshelved.

Interation	Description
	

In addition, selecting a row within the table shows some additional interaction options.



The **Shelve** button allows you to shelve the selected alarms.

The **Acknowledge** button allows you to mark the selected alarms as "acknowledged".

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
refreshRate	The rate at which the table will poll for updates in milliseconds.	value: numeric
enableHeader	Enable table header. Default is true.	value: boolean
enableDetails	Enable active events table details action. Default is true.	value: boolean
enableAcknowledge	Enable acknowledge action. Default is true.	value: boolean
enableShelve	Enable shelve action. Default is true.	value: boolean
enableUnshelve	Enable unshelve action. Default is true.	value: boolean
toolbar	Settings for the toolbar. Click to see **toolbar** properties.	object
shelvingTimes	Available alarming shelving times in seconds. Shelving times are customizable by editing values for this property in the Property Editor.	array
responsive	Responsive layout configuration. Rows are converted to cards. While in responsive layout, disables or removes certain table features that are no longer applicable. • **enabled**: Enables responsive layout. Default is false. Value is boolean • **breakpoint**: Width in pixels that triggers change in responsive layout. Value is numeric. object	
filters	This is where you configure filtering properties for displaying alarm data in the Alarm Status Table. Click to see the **filter** properties.	object

Name	Description	Property Type
selection	Currently selected alarms and alarm selection configuration. • **active**: Active alarm selection configuration and read-only list of currently selected active alarms. • **mode**: Active alarm selection configuration. Options are multiple, single, or none. Value is string. • **data**: A read-only list of currently selected active alarms. Value is array. • **shelved**: Shelved alarm configuration and read-only list of currently selected shelved alarms. • **mode**: Shelved alarm selection configuration. Options are multiple, single, or none. Value is string • **data**: A read-only list of currently selected active alarms. Value is array.	object
rowStyles	Styles to apply to rows given their alarm state and designated priority. Click to see the **rowStyles** properties.	object
activeSortOrder	The default weighted order in which columns and their contents are sorted relative to other columns and their contents. Used when the component loads. Active event columns need to have sort configured in order for this to work.	array
shelvedSortOrder	The default weighted order in which columns and their contents are sorted relative to other columns and their contents. Used when the component loads. Shelved event columns need to have sort configured in order for this to work.	array
dragOrderable	When enabled, users may drag column headers to reorder columns in the table.	value: boolean
columns	Used only for determining what columns to show on load. Click to see **columns** properties	object

Name	Description	Property Type
columnsAssociated	A list of columns used to retrieve and display alarm associated data. Click to see **columnAssociated** properties	object
pager	Settings for the pager. Click to see **pager** properties	object
style	Sets a style that applies to the component. The **Style menu** contains all the tools for modifying text, background, margins, and borders. You can also specify a **style class**.	object

toolbar Properties

Name	Description	Property Type
enabled	Enables the visibility of the table toolbar. Default is true.	value: boolean
enableActiveTab	Enables the visibility of the Active Events tab.	value: boolean
enableShelvedTab	Enables the visibility of the Shelved Events tab.	value: boolean
enableFilter	Enables the visibility of the text filter toggle. Default is true.	value: boolean
toggleableFilter	If false, the text filter will not require user interaction to open, and instead will remain open. Default is true.	value: boolean
enableFilterResults	Enables the visibility of the filters results count message. Default is true.	value: boolean
enablePreFilters	Enables the visibility of the pre-filter toggle. Default is true.	value: boolean
enableConfiguration	Enables the visibility of the configuration toggle. Default is true.	value: boolean

filter Properties

active

Settings for active alarms.

Name	Description	Property Type
text	The alarm events filter text.	value: string
states	Pre-filters for filter active alarm events. All of the following properties are boolean values. • **activeUnacked**: Active and unacknowledged. Default is true. • **activeAcked**: Active and acknowledged. Default is true. • **clearUnacked**: Cleared and unacknowledged. Default is true. • **clearAcked**: Cleared and acknowledged. Default is false.	object
priorities	Alarm event priority pre-filters. All of the following properties are boolean values. • **diagnostic**: Whether to display alarms with diagnostic priority. Default is false. • **low**: Whether to display alarms with low priority. Default is true. • **medium**: Whether to display alarms with medium priority. Default is true. • **high**: Whether to display alarms with high priority. Default is true. • **critical**: Whether to display alarms with critical priority. Default is true.	object
conditions	Gateway side alarm query conditions. All of the following properties are string values. • **source**: Filter alarms by alarm source path. Specify multiple paths by separating them with commas. Supports the wildcard *. • **displayPath**: Filter alarms by display path, falling back to the source path if a custom display path in not set. Specify multiple paths by separating them with commas. Supports the wildcard *.	object

Name	Description	Property Type
	• **provider**: Filter alarms by provider. Refer to the **Filtering Alarms** section on the Alarming page for more information on displayPath and source syntax.	
results	Alarm event filtering results configuration and data. • **enabled**: Enable filtering results to be written back to props. Value is a boolean. **Note**: Enabling this property may cause a performance decline. • **data**: An array of objects representing the current filtered data, if enabled and active.	object

shelved

Temporarily silence an alarm for a fixed period of time while the alarm event issue is worked on.

Name	Description	Property Type
text	The alarm events filter text.	value: string
results	Shelved alarm filtering results configuration and data. • **enabled**: Enable filter results to be written back to props. **Caution**: Doing so may cause performance decline. Default is false. Value is boolean. • **data**: An array of objects representing the current filtered data if enabled and active. Value is array.	object

rowStyles Properties

Name	Description	Property Type
activeUnacked	Style settings for rows with activeUnacked alarms.	object
activeAcked	Style settings for rows with activeAcked alarms.	object
clearUnacked	Style settings for rows with clearUnacked alarms.	object
clearAcked	Style settings for rows with clearAcked alarms.	object
dateFormat	A date format string to be applied against dates.	value: string

rowStyles Child Properties

Each rowStyles property except dateFormat has the following child properties. For each, a full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**

Name	Description	Property Type
base	Base style settings for alarms.	object
priorities	Style settings for the alarm row based on priority. • **diagnostic**: Style for acked alarms with diagnostic priority. • **low**: Style for acked alarms with low priority. • **medium**: Style for acked alarms with medium priority. • **high**: Style for acked alarms with high priority. • **critical**: Style for acked alarms with critical priority.	object

columns Properties

active

Active alarm event columns to display on load

Name	Description	Property Type
activeTime	Settings for the activeTime column.	object
displayPath	Settings for the displayPath column.	object
priority	Settings for the priority column.	object
state	Settings for the state column.	object
source	Settings for the source column.	object
label	Settings for the label column.	object
name	Settings for the name column.	object
eventId	Settings for the eventId column.	object
eventValue	Settings for the eventValue column.	object
notes	Settings for the notes column.	object
isActive	Settings for the isActive column.	object
isAcked	Settings for the isAcked column.	object
isClear	Settings for the isClear column.	object
ackTime	Settings for the ackTime column.	object
ackUser	Settings for the ackUser column.	object
ackNotes	Settings for the ackNotes column.	object
ackPipeline	Settings for the ackPipeline column.	object
activePipeline	Settings for the activePipeline column.	object
clearTime	Settings for the clearTime column.	object
clearPipeline	Settings for the clearPipeline column.	object

Name	Description	Property Type
deadband	Settings for the deadband column.	object

shelved

Name	Description	Property Type
expires	Settings for the expires column.	object
shelvedBy	Settings for the shelvedBy column.	object
sourcePath	Settings for the sourcePath expires column.	object

column active and shelved Child Properties

Each column type property has the following child properties.

Name	Description	Property Type
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string

columnsAssociated active Properties

Active alarm event associated data columns to display on load.

Name	Description	Property Type
field	Maps to the associated data value represented by the column.	value: string
enabled	Whether the column is enabled. Default is true.	value: boolean
width	The column's width, which when not strict represents a proportion of the available space, i.e., flex grow. If strictWidth is enabled, the column will be fixed and static.	value: numeric
strictWidth	If enabled, the width of the column (set with the width property) becomes static. Default is false.	value: boolean
sort	Default sort order of the column. Options are none, ascending, or descending.	value: string
order	Order to display this column in the table.	value: numeric

pager Properties

Name	Description	Property Type
enabled	Enables the pager to be displayed. Default is true.	value: boolean
hide	Visually hides the pager from view. Useful when pager is manipulated in a controlled fashion via the activePage property. Default is false.	value: boolean
options	Rows to show per pager option.	array
activeOption	The option to use for the table. It must exist as an available option. If the table changes as a result of user interaction, the table will write to this property internally and reflect the user's selections. External writes to this property are also possible, and will update the table accordingly. **Changed in 8.3.10** The `activeOption` property deprecates the `initialOption` property.	value: numeric
activePage	Represents the current active page and corresponds to the value of the page jump input field.	value: numeric
shelvedPage	Represents the current shelved page and corresponds to the value of the page jump input field.	value: numeric

Scripting

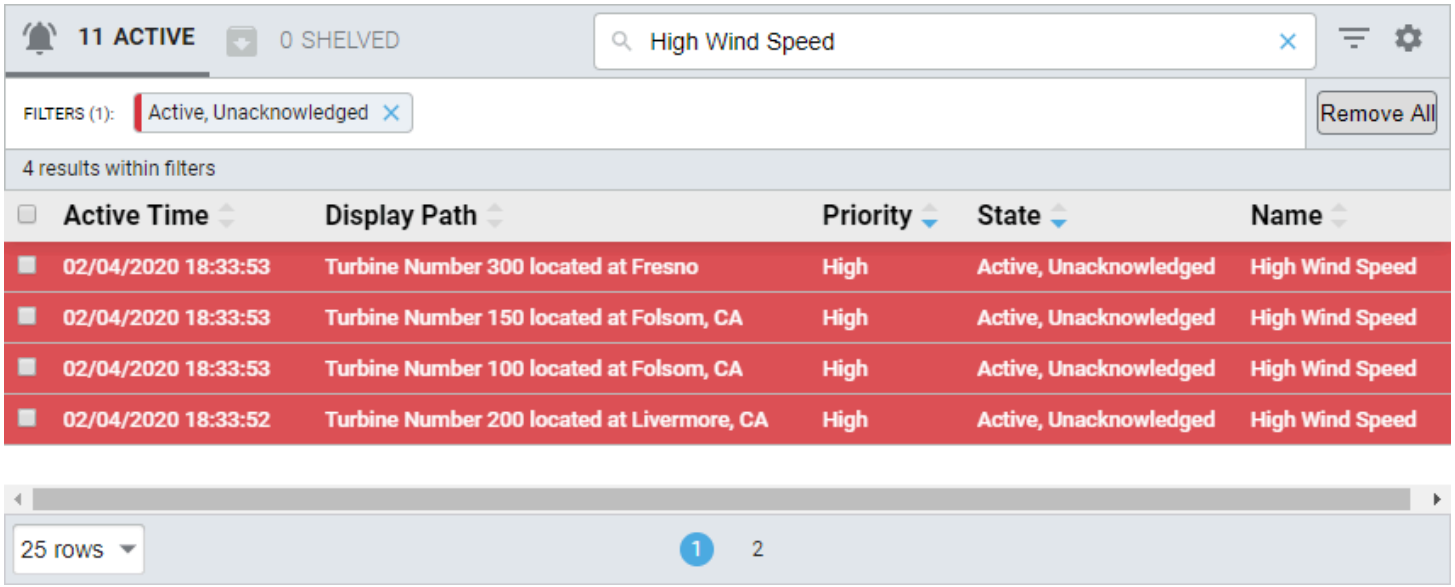
See the **Perspective - Alarm Status Table Scripting** page for the full list of scripting functions available for this component.

Examples

Example 1

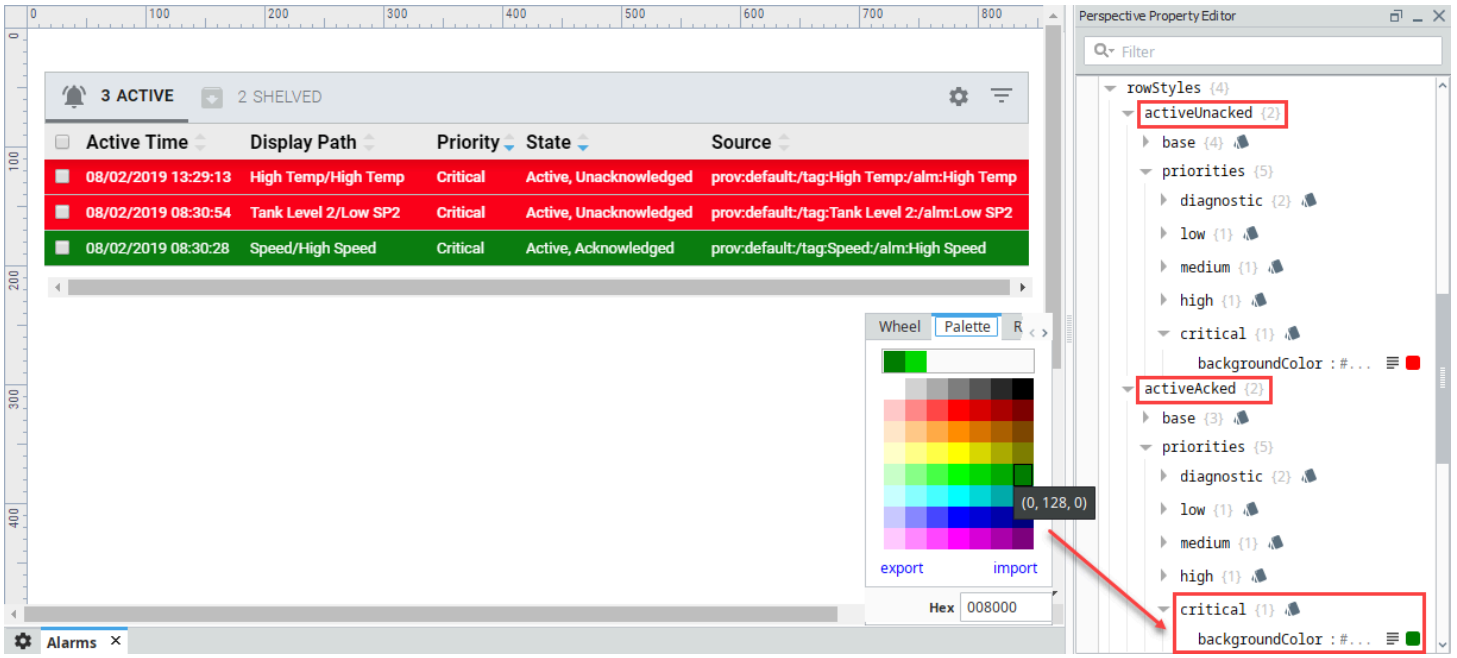
In a Perspective Session, click on the **Filter** button to filter on alarm states and/or use the **Search Bar** to target more specific alarm events. Select from the **Configuration Settings** to change the column headers to display the alarm event data you're interested in.

This example uses the Search Bar to find any text with 'High Wind Speed' in the Active, Unacknowledge alarm state.



Example 2 - Alarm Status Table Row Styles

In the Designer, you can change row styles to be different colors for the different priorities for each alarm state. In this example, the rowStyle for the Critical priority for the activeAcked alarm state was changed to green.


3 ACTIVE
2 SHELVED
Active Time
Display Path
Priority
State
Source
08/02/2019 13:29:13
High Temp/High Temp
Critical
Active, Unacknowledged
prov:default:/tag:High Temp:/alm:High Temp
08/02/2019 08:30:54
Tank Level 2/Low SP2
Critical
Active, Unacknowledged
prov:default:/tag:Tank Level 2:/alm:Low SP2
08/02/2019 08:30:28
Speed/High Speed
Critical
Active, Acknowledged
prov:default:/tag:Speed:/alm:High Speed
Wheel
Palette
(0, 128, 0)
export
import
Hex
008000
Alarms
Perspective Property Editor
Filter
rowStyles {4}
activeUnacked {2}
base {4}
priorities {5}
diagnostic {2}
low {1}
medium {1}
high {1}
critical {1}
backgroundColor : #...
activeAcked {2}
base {3}
priorities {5}
diagnostic {2}
low {1}
medium {1}
high {1}
critical {1}
backgroundColor : #...

Perspective - Alarm Status Table Scripting

This page details the various scripting, component, and extension functions available for Perspective's **Alarm Status Table** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Functions

.refreshData()

- Description
  - Refreshes the data on the Alarm Status Table.
- Parameters
  - None
- Return
  - Nothing

Extension Functions

.filterAlarm

- Description
  - Called for each event before it is displayed in the table, allowing you to hide or show each alarm event (row) in the table. Provides an opportunity to write a more complex filter than what's normally provided to the component. Return False to exclude an alarm event from the table.

- Parameters
  - ComponentModelScriptWrapper.SafetyWrapper **self**- A reference to the component that is invoking this function.
  - PyAlarmEvent **alarmEvent** - The alarm event itself. Call `alarmEvent.get('propertyName')` to inspect properties on the event. Common properties: `name`, `source`, `priority`.
- Return
  - Boolean - The function must return either a True or False for every alarm event in the table. True will show the alarm. False will hide the alarm.

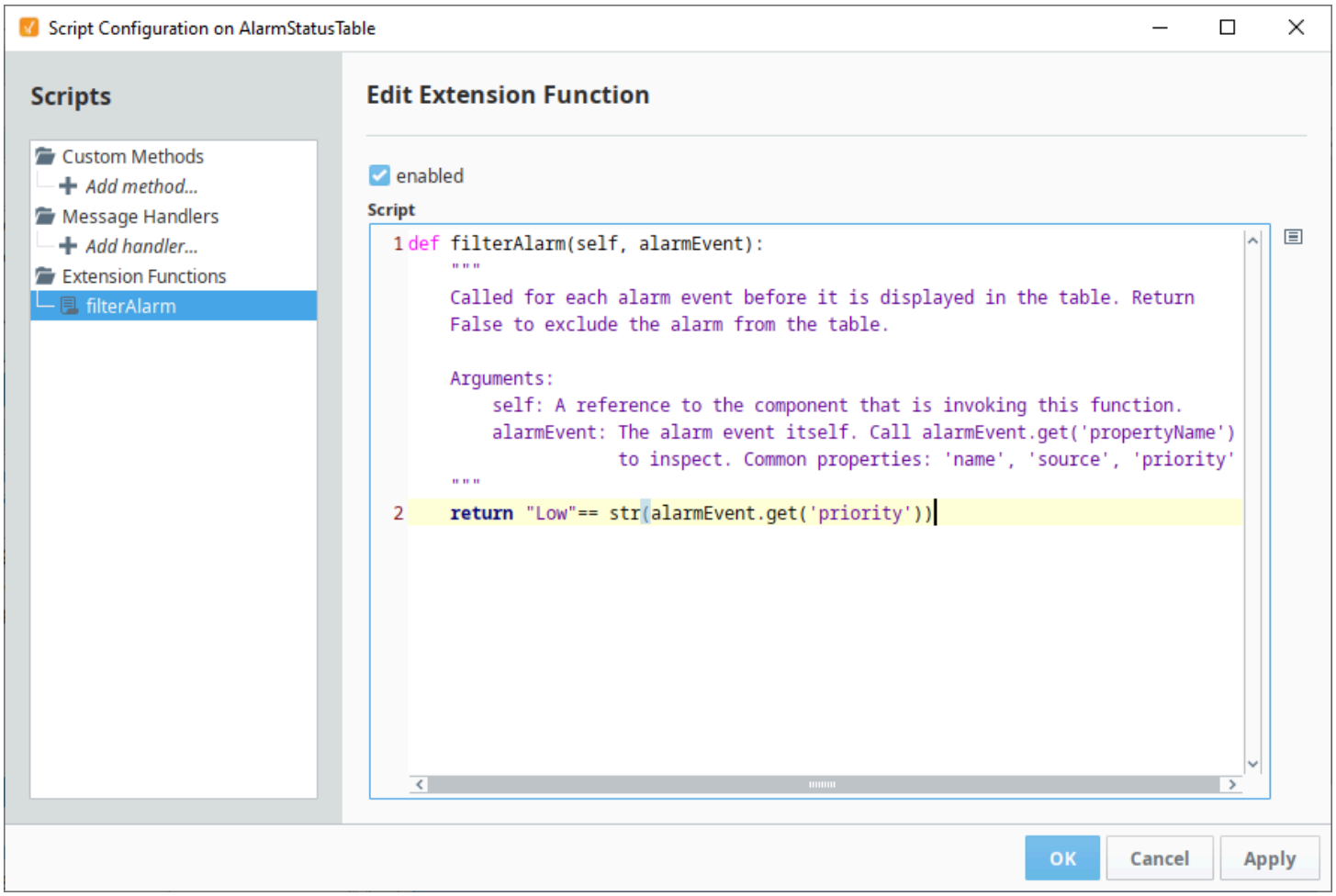
Examples

With the built-in PyAlarmEvent object all **alarm event properties** are accessible to this function, and can be used to help determine if any given event should appear on the table. Furthermore, Associated Data (also known as custom alarm properties) can be examined from the same event.

```
# Replace Property Name below with a the name of the property you wish to filter
on.
if alarmEvent.get('Property Name'):
    return True
# It's important that you return a False value for the events you don't want to
see in the table.
else:
    return False
```

You could also condense the code example above by using something like the following:

```
return "Low"== str(alarmEvent.get('priority'))
```



.filterShelvedAlarm

- Description
  - Called for each event before it is displayed in the shelved tab of the table, allowing you to hide or show each alarm event (row) in the table. Return False to exclude a shelved alarm event from the table.
- Parameters
  - ComponentModelScriptWrapper.SafetyWrapper **self**- A reference to the component that is invoking this function.
  - PyAlarmEvent **shelvedAlarmEvent** - The shelved alarm event. Call `shelvedAlarmEvent.get('propertyName')` to inspect properties on the event. Properties: `sourcePath`, `shelvedBy`, `expires`.
- Return
  - Boolean - The function must return either a True or False for every alarm event in the table. True will show the alarm. False will hide the alarm.

Example

Unlike the alarmEvent object, the shelvedAlarmEvent object may only inspect shelved alarm event properties.

```
# display only shelved alarms from a specific sourcePath:

source = shelvedAlarmEvent.get('sourcePath')
if source == 'prov:default:/tag:myTag/Mode:/alm:myAlarm':
    return True
return False
```

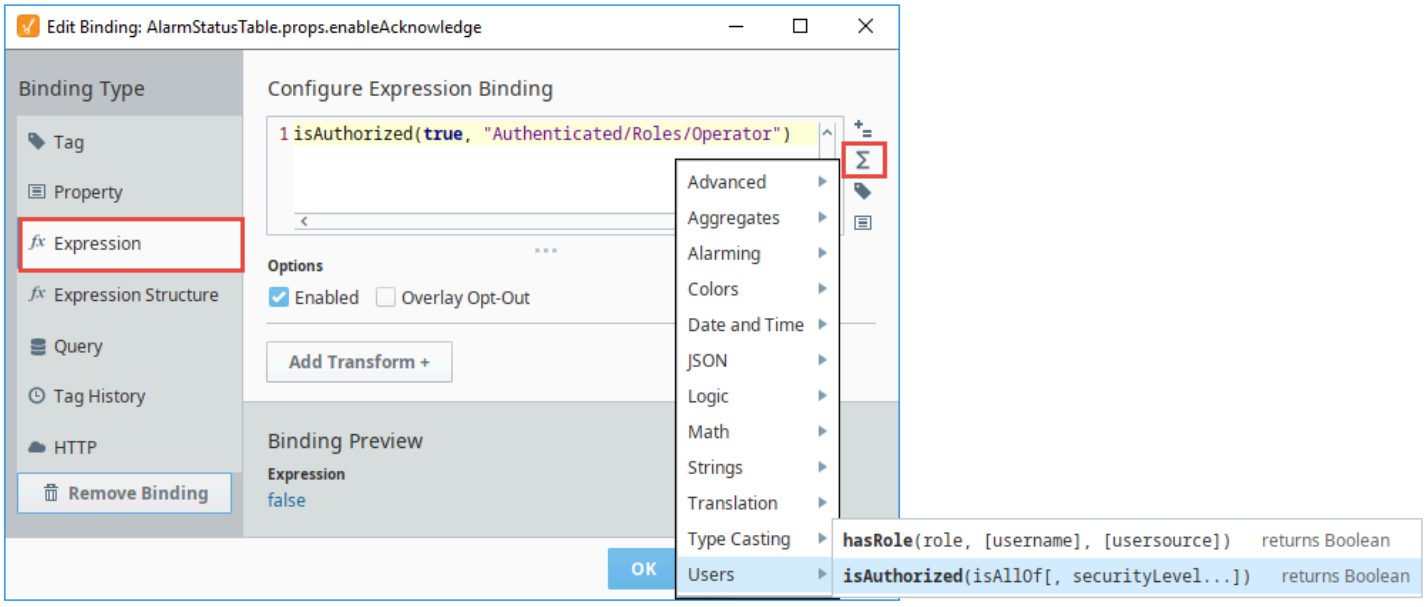
How To Restrict Acknowledgement on the Perspective Alarm Status Table

Security for Alarm Acknowledgement

You can restrict specific users or roles from Acknowledging alarms by setting the **enableAcknowledge** property in the Property Editor to **false**. This hides the Acknowledge button on the Alarm Status Table for those users who do not have permission. You can setup permissions for any **role**, user and **user source** in your system.

For example, if you only want those users with the Operator role to acknowledge alarms, the correct permission must be assigned.

1. Select the Alarm Status Table component, and click the **enableAcknowledge binding** icon to open the Property Binding window.
2. Under **Property Binding Type**, select **Expression**.
3. Click the **Function** Σ icon and scroll down to **Users**, and select **isAuthorized**. This enters the function name.
4. Edit the expression to read: **isAuthorized(true, "Authenticated/Roles/Operator")**.
5. Click **OK**.



If you currently have the 'Operator' role, you'll notice in the Property Editor of the Designer that the **enableAcknowledge** property is set to **true**, and for other roles, it will be set to **false**.

Perspective - Audio



Component Palette Icon:



> **INFO**
>
> The UI for this component is hidden by default and is browser dependent. There is a hidden property, `props.display`, which can be used to display UI for this component. In the event a user is displaying the UI (`props.display = true`), the component will be displayed differently based on the browser in use.

Description

An audio component, hidden by default, that Designers can use to play and pause sound clips in the browser.

Supported Audio Files

Supported audio file types are browser dependent.

File Type	Browser Compatibility
3GP	Firefox for Android
ADTS	Firefox Available only if available on the underlying operating system's media framework.
FLAC	Chrome 56, Edge 16, Firefox 51, Safari 11
MPEG-4 (MP4)	Chrome 3, Edge 12, Firefox, Internet Explorer 9, Opera 24, Safari 3.1
Ogg	Chrome 3, Firefox 3.5, Edge 17 (desktop only), Internet Explorer 9, Opera 10.50 Edge requires Web Media Extensions to be installed.
QuickTime (MOV)	Only older versions of Safari, plus other browsers that supported Apple's QuickTime plugin
Waveform Audiofile (WAV)	Chrome 8+, Edge 12+, Firefox 4+, Opera 11.5+, Safari 4+
WebM	Chrome 6, Edge 17 (desktop only), Firefox 4, Opera 10.6, Safari 14.1 (macOS), Safari 15 (iOS). Edge requires Web Media Extensions to be installed.

Properties

Most Properties have binding options. For more information on Bindings, see Types of Bindings in Perspective. This section only documents the Props Category of properties. The other Categories are described on the Perspective Component Properties page.

Name	Description	Property Type
source	The source URL of the media file.	value: string
play	The play state of the media file. Toggling this property will start or pause the media file.	value: boolean
loop	Determines if the media file should loop after reaching the end.	value: boolean
volume	The percentage of maximum volume (from 0 to 100).	value: numeric
playbackRate	A double that represents the playback rate of the media file.	value: numeric
allowDownload	Determines whether the audio player allows downloading of the media file.	value: boolean
style	Sets a style that applies to the component. The **Style menu** contains all the tools for modifying text, background, margins, and borders. You can also specify a **style class**.	object

Scripting

See the **Perspective - Audio Scripting** page for the full list of scripting functions available for this component.

Perspective - Audio Scripting

This page details the various scripting, component, and extension functions available for Perspective's **Audio** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onPlay

This event is fired when playback has begun.

event

- Object Path
  - event
- Type
  - Null
- Description
  - An empty event object.

onPause

This event is fired when playback has been paused.

event

- Object Path
  - event

- Type
  - Null
- Description
  - An empty event object.

onError

This event is fired when there is an error attempting to play the media file.

> **CAUTION**
> This component event is designed to be used in tandem with a script action. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.errorMessage

- Object Path
  - event.errorMessage
- Type
  - String
- Description
  - Error message when attempting to play the media file.

onEnded

This event is fired when playback has ended due to reaching the end of the media.

event

- Object Path
  - event
- Type

    - Null
- Description
    - An empty event object.

onLoaded

This event is fired when the first frame of the media has loaded.

event

- Object Path
    - event
- Type
    - Null
- Description
    - An empty event object.

onRateChanged

This event is fired when the playback rate of the media has changed.

event

- Object Path
    - event
- Type
    - Null
- Description
    - An empty event object.

Component Functions

.play()

- Description
  - Plays the media file, triggering the onPlay component event.
- Parameters
  - None
- Return
  - Nothing

.pause()

- Description
  - Pauses the media file, triggering the onPause component event.
- Parameters
  - None
- Return
  - Nothing

.replay()

- Description
  - Replays the media file from the beginning.
- Parameters
  - None
- Return
  - Nothing

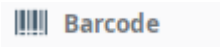
Extension Functions

This component does not have extension functions associated with it.

Perspective - Barcode



Component Palette Icon:



Description

The Barcode component enables you to display text as a barcode. The component supports 105 different barcode types including Code 128, QR code, EAN-8, and ISBN.

The Barcode component has two pre-configured **variants**: Code 128 and QR Code.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	Value to be encoded as a barcode.	value: numeric
type	What barcode specification to use, currently there are 105 formats supported including Code 128, QR code, EAN-8, and ISBN. Click here to see the **barcode specification options**.	value: string
displayValue	If true, the barcode's value will be displayed as text.	value: boolean
valuePosition	If displayValue is true, this property determines where the value should be displayed. Options are top or bottom; default is bottom.	value: string
valueStyle	Sets a style for the display value for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
errorStyle	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
style	Sets an overall style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Barcode Specification Options

A-B	C	D	E-G
auspost azteccode azteccodecompact aztecrune bc412	channelcode, codablockf, code11 code128 code16k code2of5 code32 code39 code39ext code49 code93 code93ext codeone coop2of5	daft databarexpanded databarexpandedcomposite databarexpandedstacked databarexpandedstackedcomposite databarlimited databarlimitedcomposite databaromni databaromnicomposite databarstacked databarstackedcomposite databarstackedomni databarstackedomnicomposite databartruncated databaretruncstedcomposite datalogic2of5 datamatrix datamatrixrectangular datamatrixrectangularextension dotcode	ean13 ean13composite ean14 ean2 ean5 ean8 ean8composite flattermarken gs1-128 GS1-128 gs1-128composite gs1-cc gs1datamatrix gs1datamatrixrectang gs1northamericancou gs1qrcode

H-J	K-P	Q-Z
hanxin hibcazteccode hibccodeablockf hibccode128 hibccode39 hibcdatamatrix hibcdatamatrixrectangular hibcmicropdf417 hibcpdf417 hibcqrcode iata2of5 identcode industrial2of5 interleaved2of5 isbn ismn issn itf14 japanpost	kix leitcode mailmark matrix2of5 maxicode micropdf417 microqrcode msi onecode pdf417 pdf417compact pharmacode pharmacode2 planet plessey posicode postnet pzn	qrcode rationalizedCodabar raw royalmail sscc18 symbol telepen telepennumeric ultracode upca upcacomposite upce upcecomposite

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1



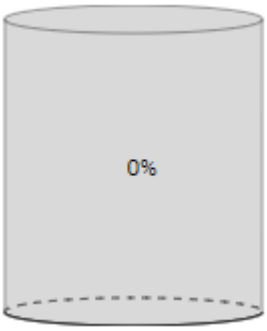
Property	Value
props.value	014113910613
props.type	upca
props.displayValue	true
props.valuePosition	bottom
props.valueStyle.fontFamily	Verdana
props.valueStyle.fontSize	18px

Example 2



Property	Value
props.value	http://inductiveautomation.com
props.type	qrcode
props.displayValue	true
props.valuePosition	top
props.valueStyle.color	#2747C7
props.valueStyle.fontFamily	sans-serif
props.valueStyle.fontSize	14px
props.valueStyle.fontWeight	bold
props.style.paddingTop	12px
props.style.borderColor	D97700

Perspective - Cylindrical Tank



Component Palette Icon:



Description

This component looks like a 3D cylindrical tank with liquid inside. It can be configured so that the "liquid" rises and falls as the `value` property changes. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	Numeric value of the tank's level.	value: numeric
capacity	Total capacity of the tank. Default is 100.	value: numeric
liquidColor	Color used to render the filled part of the tank. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector.**	color
tankColor	Color of the non-filled tank section. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector.**	color
liquidOpacity	The opacity of the liquid in the tank. 0 is fully transparent, 1 is fully opaque. Default is 0.5.	value: numeric
liquidWarningColor	The warning color of the liquid in the tank. See **Color Selector.**	color
tankWarningColor	The tank warning color. See **Color Selector.**	color
warningThreshold	The warning appearance will be used when value as a percentage of the capacity exceeds this value. Default is 100	value: numeric
strokeWidth	The stroke width, in pixels, for the outside of the tank. Default is 1.	value: numeric
valueDisplay	Value display configuration. Renders and styles a value overlay in the tank. Click here to see **valueDisplay** properties.	object
style	Sets a style for this cylindrical tank. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

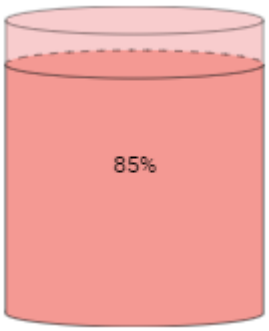
valueDisplay Properties

Name	Description	Property Type
enabled	Whether valueDisplay is shown. Default is true.	value: boolean
style	Modify the valueDisplay style using the style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
format	Format to apply to value which is then used as the display value. Available options include: • None • Integer • Percent • Currency	value: string
unit	Unit value to display on value overlay: • **enabled**: If true, will show either a prefixed or postfixed unit. Value is boolean. • **value**: Unit value to display. Value is string. • **fix**: Direction in which to place the unit. Either as a prefix or a postfix. Value is string.	object

Component Events

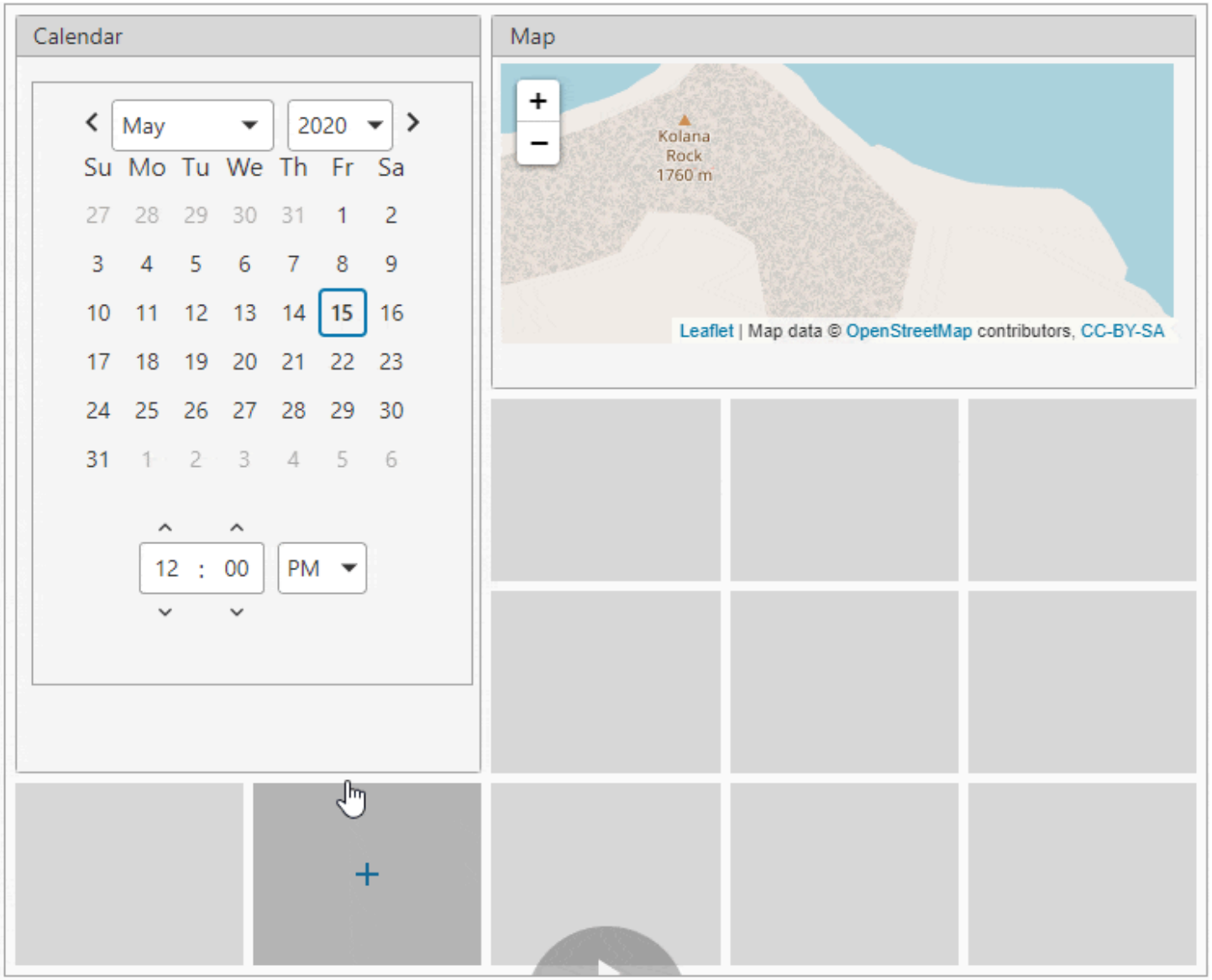
The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example



Property	Value
props.value	85
props.capacity	100
props.warningThreshold	80

Perspective - Dashboard



Component Palette Icon:



Description

The Dashboard component exposes layout capabilities to end users in a Perspective Session so they have the ability to customize their dashboard layout for their individual needs. Widgets are configured in the Designer by designers and made available to Perspective Session users. The Dashboard component uses a grid system based off of CSS grid specifications to position and place widgets.The Property Editor of the Dashboard component is where the designer controls the general layout of the grid by specifying the responsive mode: fixed or stretch, if the dashboard is editable, and if each widget is configurable and available in a Perspective Session.

End users can choose from a list of pre-configured widgets to configure their dashboards in a Perspective Session. They can add, remove, resize, move around, and configure widgets, including the ability to interact with widgets in a Session such as entering text in a text field, displaying/hiding components in a widget, and even use parameters to pass a property to a specified view.

To learn more, refer to **Configuring a Dashboard**.

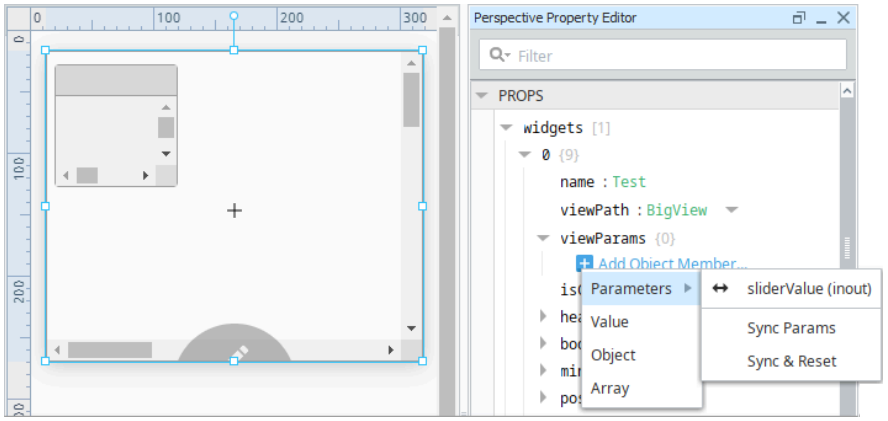
Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
pack	Enables widget packing algorithm. When disabled, widgets can be placed anywhere on the Dashboard and the component will not try to rearrange them in an optimal layout.	boolean
grid	The grid layout mode defines the responsive behavior of the grid and its cells: fixed and stretch. Both fixed and stretch values are strings. • **fixed**: In fixed mode, the grid's dimensions can be greater or less than the full dimensions of its containing element, and its cells are given a static size, effectively creating a scrollable grid when cells overflow beyond the containing elements dimensions. • **stretch**: In stretch mode, the grid's dimensions are restricted to the full dimensions of its containing element, and its cells consume one free unit of space, effectively growing and shrinking with the containing element.	object
isEditing	Controls the runtime edit mode of the dashboard component. Stays in sync with the edit/play toggle control located at the bottom of the component.	boolean
editingToggle	Whether to display the dashboard editing toggle option. When disabled, hides the built in edit/play toggle control located at the bottom of the component. Disable this if you'd like to implement your own toggle that updates the `isEditing` prop in a controlled fashion. Default is true.	boolean
fixed	Visible when the grid mode is `fixed`. The following properties are all numeric values. • **cellSize**: Width and height of a grid cell. Exclusively for fixed mode. • **rowCount**: The number of rows in the grid. • **columnCount**: The number of columns in the grid. • **rowGuttersize**: The gap size between grid rows. • **columnGutterSize**: The gap size between grid columns.	object

Name	Description	Property Type
stretch	Visible when the grid mode is `stretch`. The following properties are all numeric values. • **rowCount**: The number of rows in the grid. • **columnCount**: The number of columns in the grid. • **rowGuttersize**: The gap size between grid rows. • **columnGutterSize**: The gap size between grid columns.	object
widgets	An array of configuration objects for widgets currently in use the dashboard display. **Click here** to see widgets properties.	array
availableWidgets	An array of widgets as configuration objects that are available to the user. When a widget is added to the dashboard via the add widget modal, this configuration object is copied to the widgets in use array, and act as the widgets defaults. **Click here** to see availableWidgets properties.	array

widgets Properties

Name	Description	Property Type
name	The unique widget name.	value: string
viewPath	The current configuration view path of the widget.	value: string
viewParams	Parameters for the view. If passing parameters into the embedded view, the names here must match the parameters on that view. A dropdown list of parameters is available when the user clicks the **Add Object Member** icon. This makes it easy to add parameters from the rendered view. 	object
isConfigurable	Whether this widget is configurable during runtime. If enabled, dashboard is in edit mode, the toggle becomes available when the widget is selected which is used to configure the widgets view. When toggled on, the configuring view parameter will be true.	value: boolean
header	Configuration object for the widget header. • **enabled**: When enabled, renders the widget header. Value is boolean. • **title**: The header title to display. Value is string. • **style**: Style to be applied to the widget header. A full menu of **style options** is available. You can also specify	object

Name	Description	Property Type
	a **style class**.	
body	Configuration object for the widget body. • **style**: Style to be applied the widget. A full menu of **style options** is available. You can also specify a **style class**.	object
minSize	Specifies the widgets minimum allowable size when determining widget layout. Users may not resize widgets below these dimensions. The following property values are numeric. • **columnSpan**: The minimum allowable columns that this widget may span. • **rowSpan**: The minimum allowable rows that this widget may span.	object
position	The widget position in the dashboard. Whenever a widget is added, resized, or moved the widget position object is automatically updated. The following property values are numeric. • **rowStart**: The top position of the widget. • **rowEnd**: The bottom position of the widget • **columnStart**: The left position of the widget. • **columnEnd**: The right position of the widget.	object
style	Style to be applied the widget. A full menu of **style options** is available. You can also specify a **style class**.	object

availableWidgets Properties

Name	Description	Property Type
viewPath	The current configuration view path of the widget.	string
viewParams	Parameters being passed to the view at the specified path. A dropdown list of parameters is available when the user clicks the **Add Object Member** ➕ icon. This makes it easy to add parameters from the rendered view.	object
isConfigurable	Whether this widget is configurable during runtime. If enabled and the dashboard is in edit mode, the toggle becomes available when the widget is selected which is used to configure the widgets view. When toggled on, the configuring view parameter will be 'true.'	value: boolean
defaultSize	Specifies the widgets default size adding a widget with no size specified. The following property values are numeric. • **columnSpan**: The default columns that this widget will span. • **rowSpan**: The default rows that this widget will span.	object
minSize	Specifies the widgets minimum size used when determining widget layout. The following property values are numeric. • **columnSpan**: The minimum allowable columns that this widget may span. • **rowSpan**: The minimum allowable rows that this widget may span.	object
category	A category in which to group this widget when displayed in the add widgets modal.	value: string
name	A unique name to provide this widget. This is used in the add widget modal. If no name is specified, its value will be blank. This is a required property.	value: string

Name	Description	Property Type
header	Widget header configuration. • **enabled**: Whether the widget header should show. Value is boolean. • **title**: The header title to display. Value is string. • **style**: Style to be applied to the widget header. A full menu of **style options** is available. You can also specify a **style class**.	object
body	Widget body configuration. • **style**: Style to be applied to the widget header. A full menu of **style options** is available. You can also specify a **style class**.	object
style	Style to be applied the widget. A full menu of **style options** is available. You can also specify a **style class**.	object

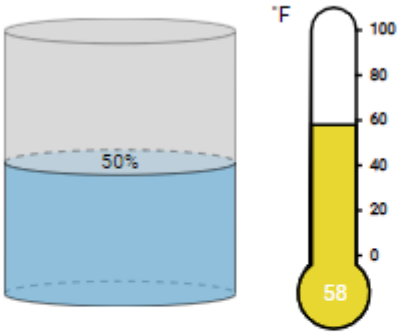
Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

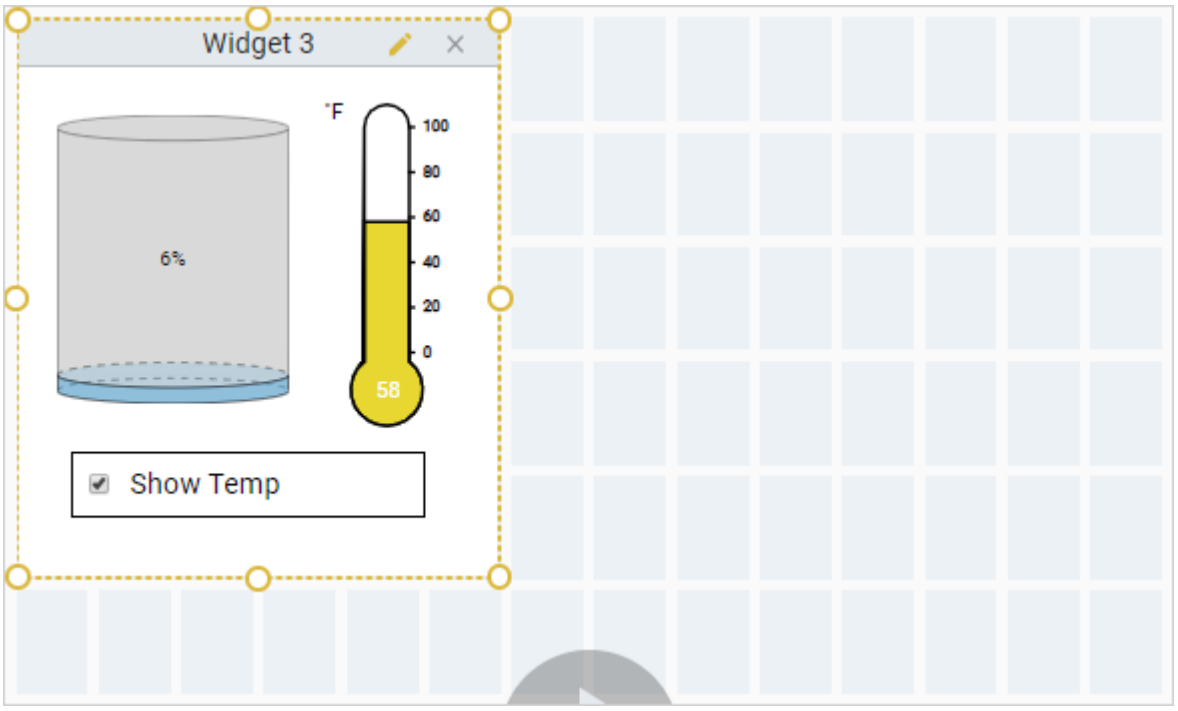
This image shows the dashboard in a Perspective Session with one widget.

Widget 3

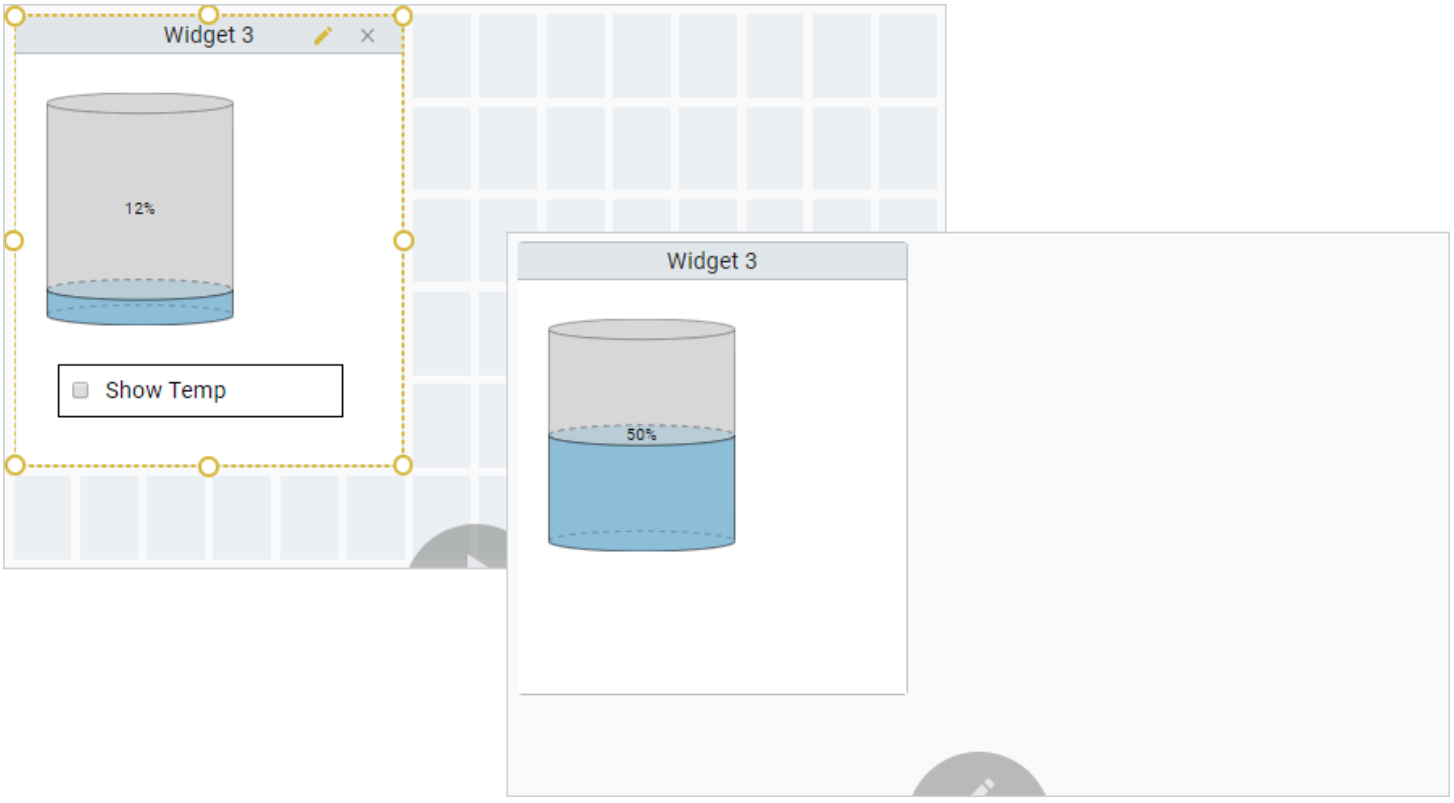


Property	Value
props.grid	stretch
props.isEditing	true
props.editingToggle	true
props.stretch.rowCount	6
props.stretch.columnCount	14
props.widgets.0.name	Widget 3
props.widgets.0.viewPath	Widget 3
props.widgets.0.isConfigurable	true
props.widgets.0.header.enabled	true
props.widgets.0.header.title	Widget 3
props.availableWidgets.2.viewPath	Widget 3
props.availableWidgets.2.isConfigurable	true
props.availableWidgets.2.name	Widget
props.availableWidgets.2.header.enabled	true
props.availableWidgets.2.header.title	Widget 3

This same example also allows the user to interact with the widget in Edit mode when the `isConfigurable` property is set to **true**. The view uses a `configuring` parameter to go into **configuring** mode allowing users to configure the widget. It allows the user to show/hide the Thermometer showing the temperature of the Tank by toggling the ShowTemp checkbox.



By unchecking Show Temp, the Thermometer component is removed from the widget on the dashboard in a Perspective Session.



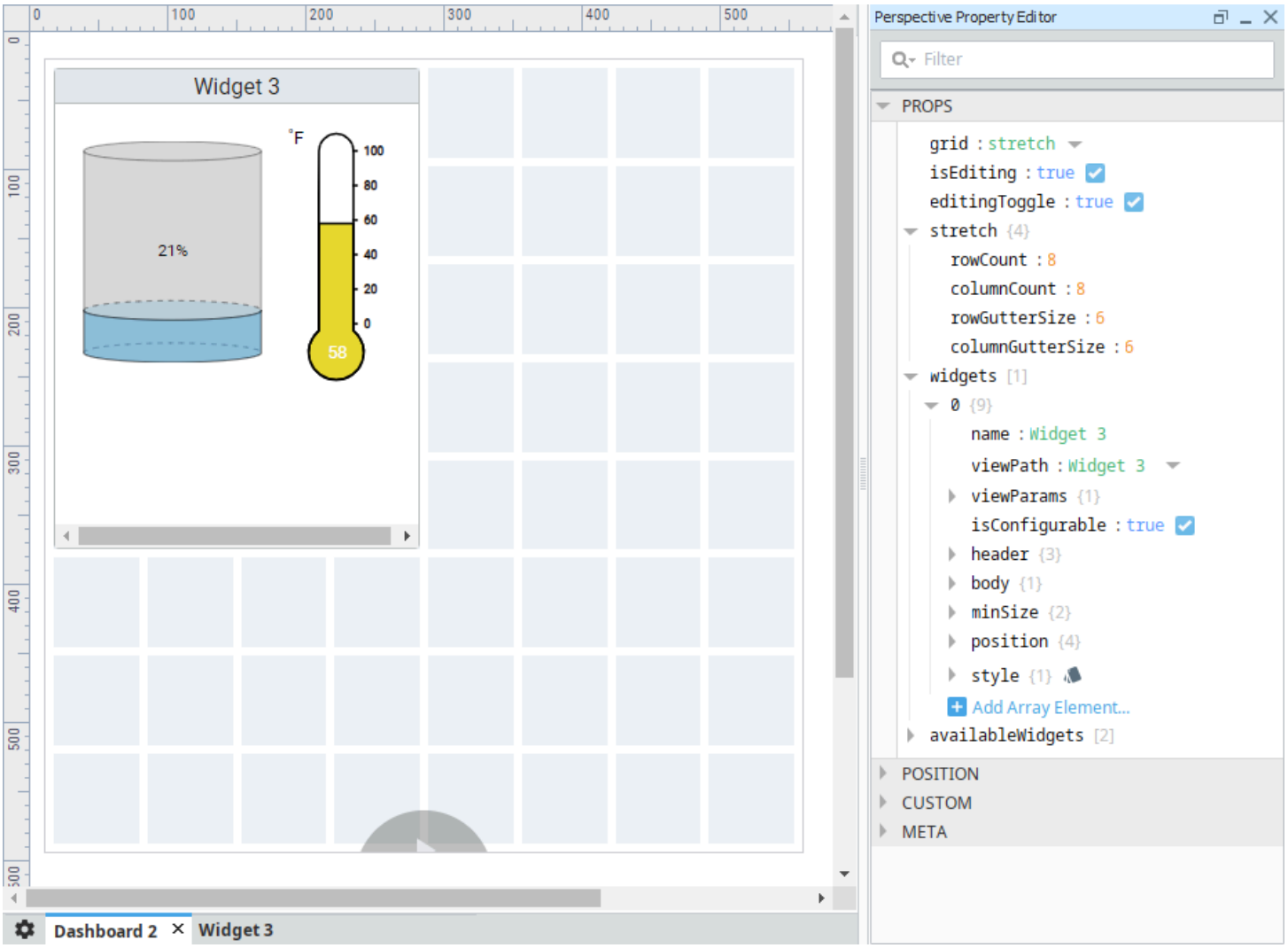
Configuring a Dashboard

The Dashboard exposes widgets to end users in a **Perspective Session** so they can customize their dashboard layout for their individual needs. Widgets are views that are pre-configured in the Designer and made available to Perspective Session users. End users have the flexibility to add, remove, resize, move around, and even configure widgets in the dashboard of their Perspective Session without having access to the Designer. Users can interact with widgets in a Session on both desktop and mobile devices. There may be some minor variances in how a user can interact with their dashboard between desktop and mobile devices, but the principle is still the same.

Configuring a Dashboard Component

Configuring a Dashboard starts with designing widgets and having a selection of pre-configured widgets for users to choose from to configure their individual dashboards. Designers create the widgets and make them available for end users to use in their individual dashboards. By making the widgets available using the `availableWidgets` property, the widget overlay modal is populated with a searchable list of all the available widgets a user can add to their dashboard. The dashboard component contains a host of additional properties that can be configured based on the end-user requirements.

The Dashboard component uses a grid system based off of CSS grid specifications to position and place your widgets. The Property Editor settings of the Dashboard component control the general layout of the grid. They specify the responsive fixed or stretch modes, if the dashboard is editable, and if each widget is configurable and available in a Perspective Session. The image below shows one widget on a dashboard in the Designer along with some of its properties.

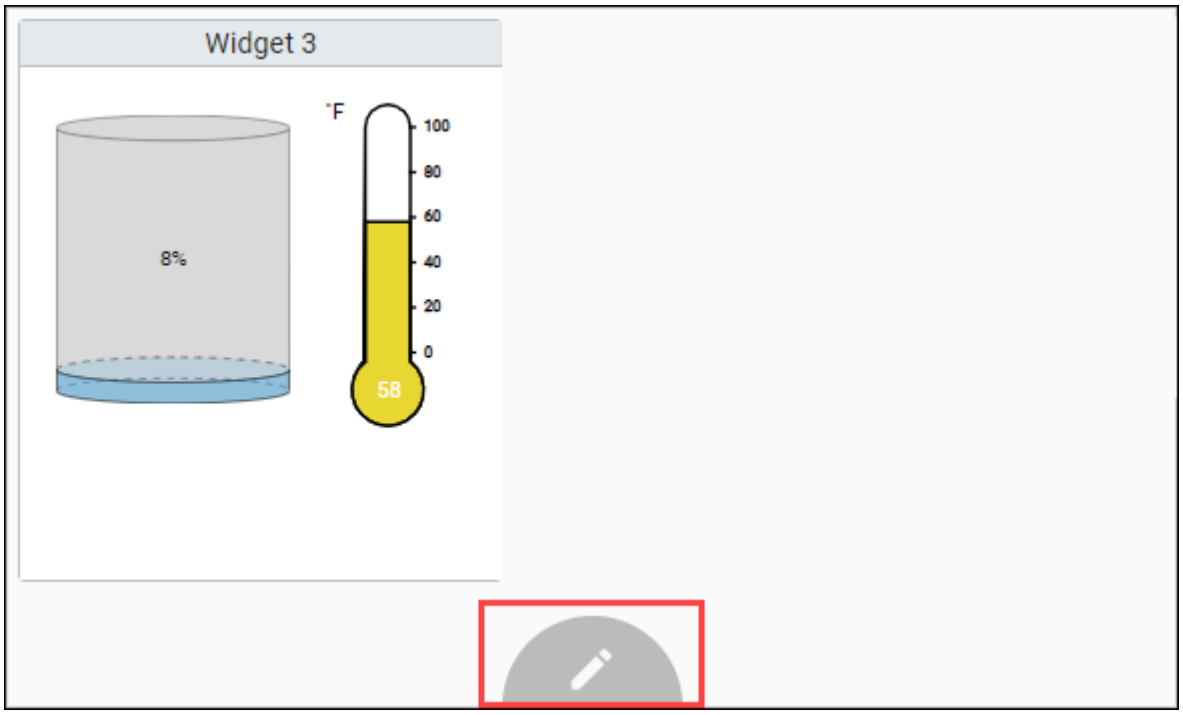


To learn more about Dashboard properties, refer to the **Dashboard component** page.

Setting Up a User Dashboard

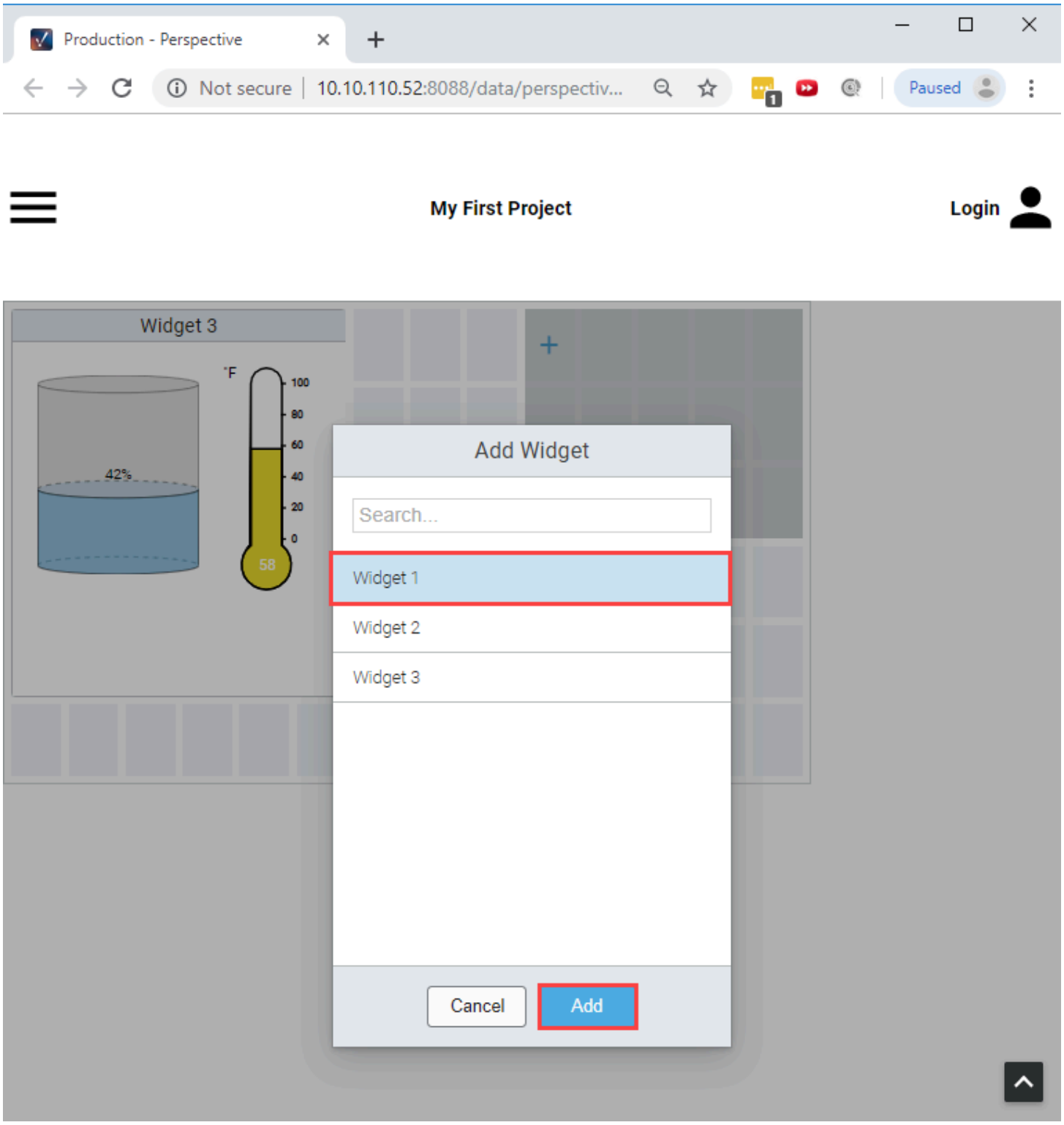
Setting up a dashboard starts with users choosing from a list of pre-configured widgets to fill their dashboards in a Perspective Session based on their individual needs.

To edit the dashboard in a Perspective Session, the user can put the dashboard into Edit mode by clicking the Edit icon at the bottom of the dashboard and deciding what widgets they want, where they want them, and how they want them configured. They can add, remove, resize and configure widgets, including the ability to interact with widgets such entering text in a text field or displaying/hiding components in a widget. You can also remove this control entirely and implement your own by configuring the `editingToggle` property on the component.



Adding a Widget

There are two ways a user can add a widget in a Perspective Session: by clicking on a single grid cell, or by dragging a grid cell over multiple grid cells that opens an add widget overlay as shown in the image below. Both ways result in displaying the add widget modal which provides a searchable list of all of the available widgets a user may add to their dashboard.



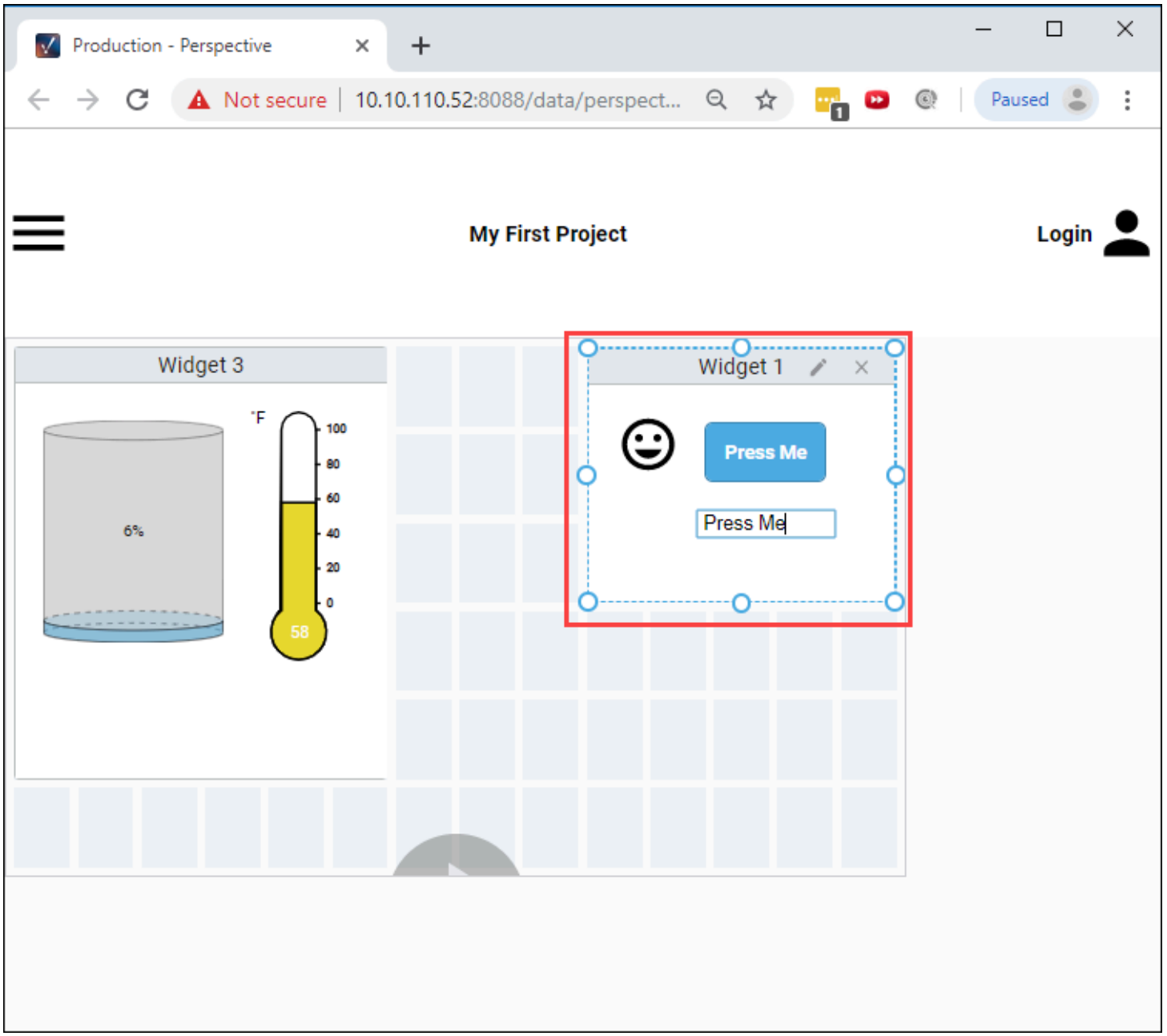
Dragging a grid cell creates an add widget overlay that specifies the desired dimensions of the widget to add. If the desired widget position overlaps other widgets, the overlapped widgets will be moved to any available space on the dashboard. Widgets do not overlap when being added, resized, and moved unless there happens to be no space for a widget so that it is placed within the grid.

WIDGET'S MINIMUM DIMENSIONS

When adding a widget, if the desired dimensions are less than the configured minimum dimensions, the desired dimensions will get overridden by the minimum dimensions. If a single grid cell is clicked, the configured default dimensions will be applied, if and only if, the default

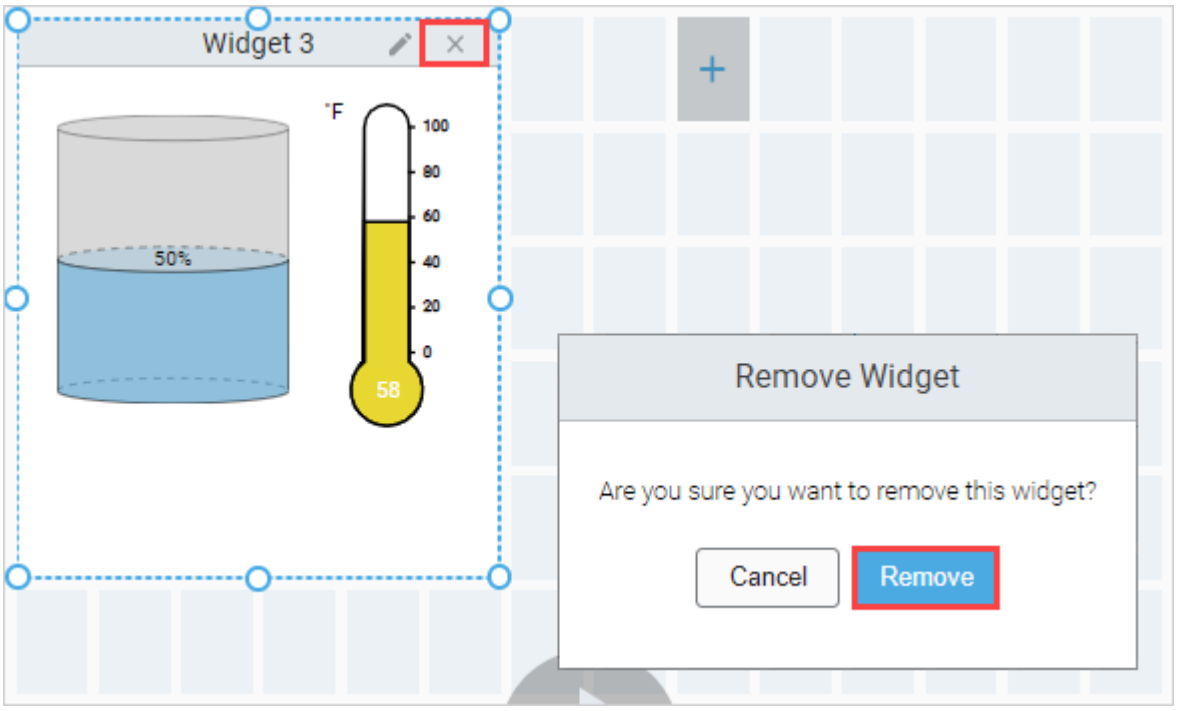
meets the required minimum dimensions, otherwise the minimum dimensions are applied. By default, the minimum and default dimensions for a widget are 1x1.

On mobile devices, activating a grid cell requires a long-press of about a second. Once a grid cell is activated, you can then drag to create the add widget overlay. The image below shows Widget 1 dropped over the multiple selected grid cells in the dashboard. You'll notice the active widget has a dashed blue border.



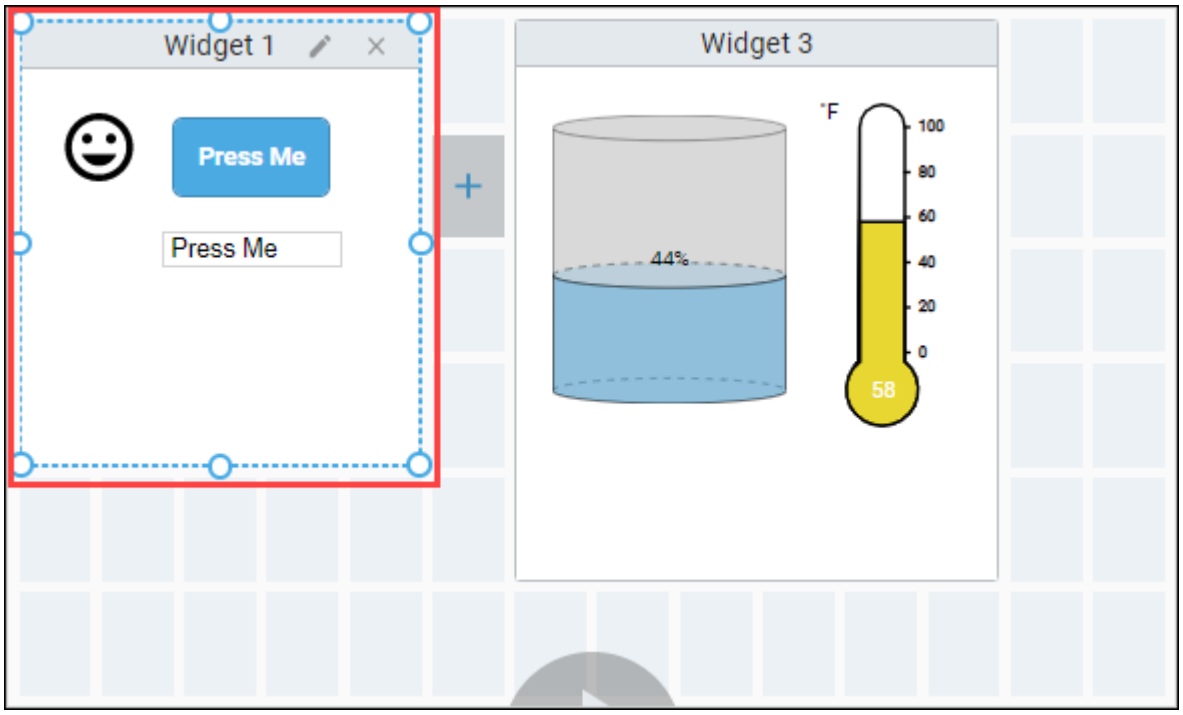
Removing a Widget

Click the **Edit** icon, select the widget, and you'll notice the widget has a dashed blue border indicating the widget is active, then click the **Delete X** icon in the top right corner to remove the widget from the dashboard. You will then be prompted with a confirmation modal to delete the widget. Click **Remove**.



Moving a Widget

Put the dashboard in Edit mode, select the widget so that it becomes active (dashed blue border). Drag the widget to the desired position. As you move the widget, any overlapped widgets will be repositioned into the first available space.



Resizing a Widget

To resize a widget, put the dashboard in Edit mode, then simply select the widget you'd like to resize and drag one of the resize handles. If, while resizing, the widget overlaps other widgets, the overlapped widgets will be repositioned into the first available space.

Configuring a Widget

The dashboard allows your users to configure a widget in a Perspective Session. To do this, you need make a few changes to your view and Dashboard component configuration.

Setting a Widget as Configurable in the Designer

To make a view allow configuration, you need to set the `isConfigurable` property for each widget that needs to be configured. This will set a param value on your view (in the runtime) that you can use to create a configuration display in your view. The purpose of this parameter is to avoid having to make a separate widgets for each possible variation of the same view.

1. Select the Dashboard component.
2. Expand the `availableWidgets` parameter, and expand the array object for the widget that you want to make configurable.
3. Set the `isConfigurable` property to **true** for this widget.

Creating a Configurable View in the Designer

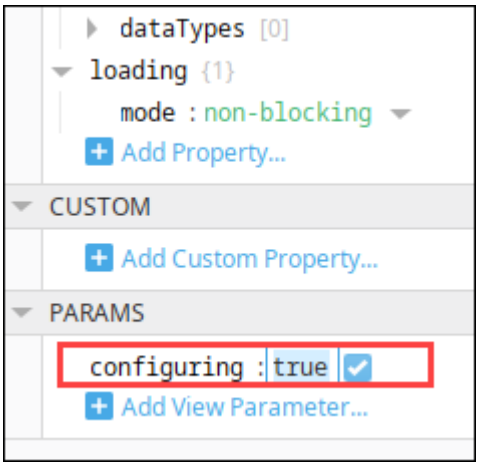
To make a view configurable, you need to do a bit of work to alter what is in the view. This is possible because the Dashboard component was created to use a parameter named `configuring` that is set to **true** when the widget is in put into configuration mode. The idea here is to have a second mode, or "display" version, of the view that has controls on it to effect the primary display. The best way to do this is to create two containers in your view; one for configuration and one for display. You can then bind the visibility on each container so only one is shown at a time.

To learn more about using parameters to pass properties, refer to the **Perspective Component Properties** page. You will not need to pass any value into the param though, it is done automatically for you if you get the param name correct.

1. Create a new Coordinate view. For the example, we named our Configurable_View.

   > **NOTE**
   >
   > If you use a Flex container, some of the settings will be different further down in the example.

2. In the Property Editor, add a view param. Name the param `configuring` and set the value to **true**. Note that all letters are lowercase.



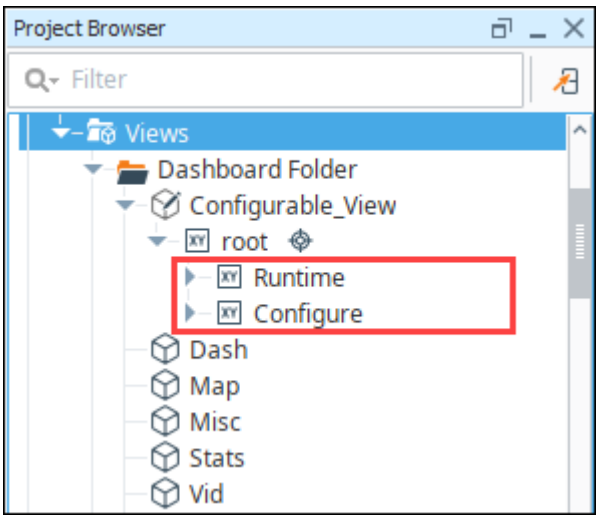
3. Drag a Coordinate container component inside your view. Name the container Runtime.

4. Add any display components you want.

> **NOTE**
>
> If you started from an existing view, move all existing components into the new container, and then make the container fill your entire view.

   i. Deep Select the Configure container.
   ii. Drag a Cylindrical Tank component into it.
   iii. Bind the `value` property to a tag.
   iv. Drag a Thermometer component into it.
   v. Bind the `value` property to a tag.

5. We need to create a second new container in the view for your configuration. **Duplicate** the Runtime container. Name the container Configure. This container will be a sibling to the Runtime container, not inside the Runtime container.



6. Add a Checkbox to toggle the Thermometer component visibility.

> **NOTE**
>
> If you started with an existing view, this step is completely up to you. You will decide what should be configurable and create controls for that in your Configure container. For example, you could create a list of tags for the user to select between, and display only the selected tags on a chart.

i. Deep Select the Configure container.

ii. Drag a Checkbox component into it.

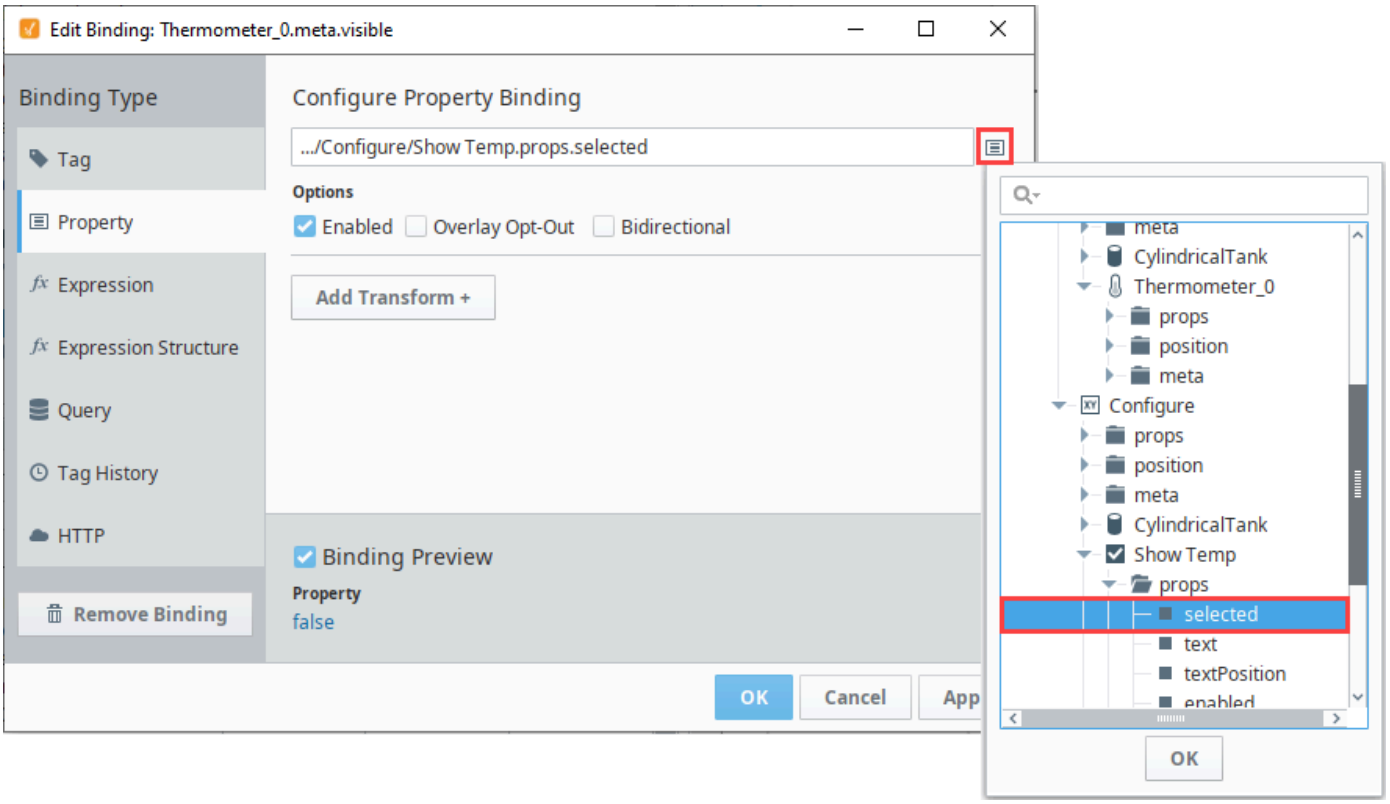
iii. Set the Name and Text to **Show Temp**.

7. Now, alter the Runtime container components to listen to our new controls.

i. Deep Select the Runtime container.

ii. Select the Thermometer component.

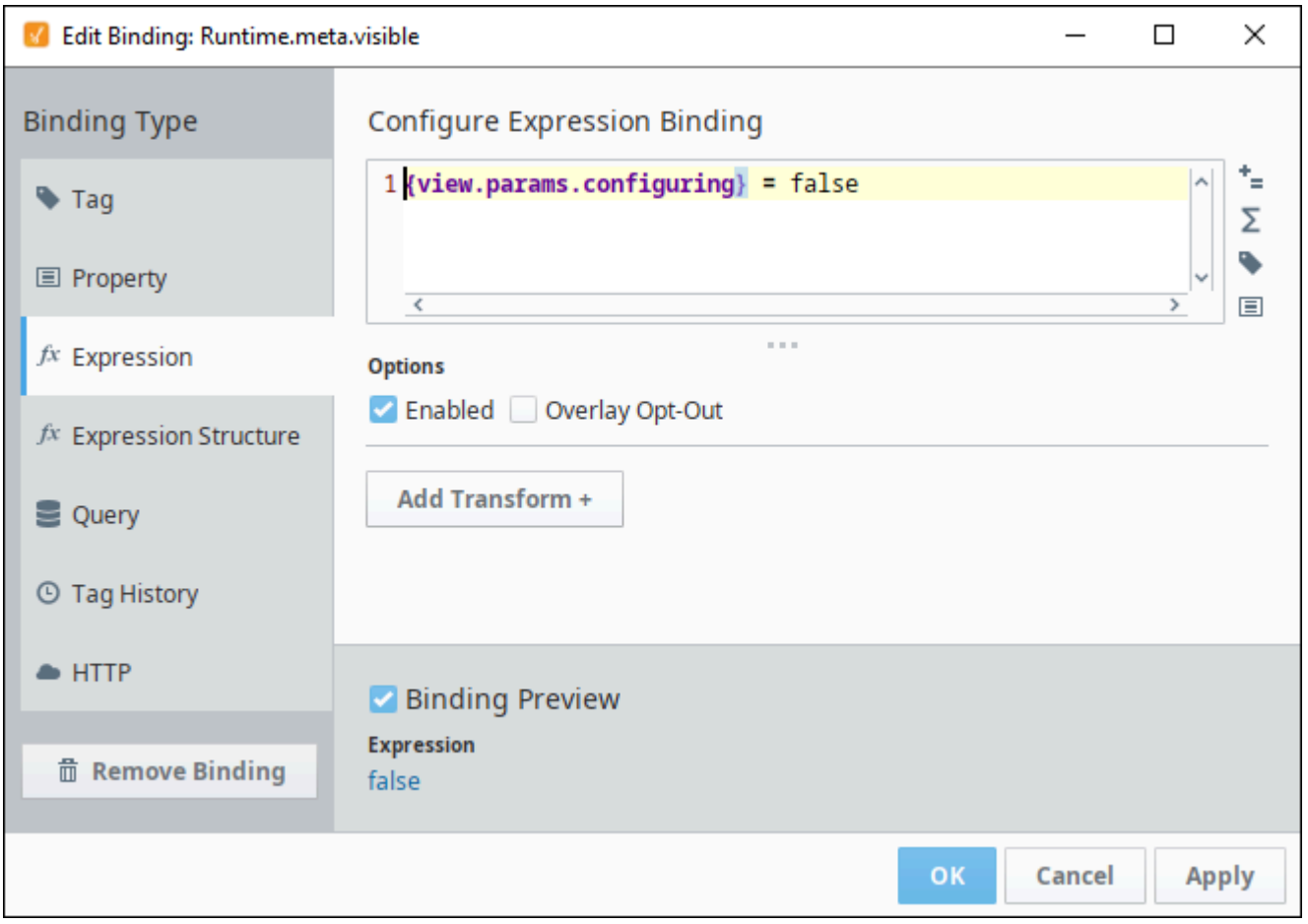
iii. In the Property Editor under META, bind the `visible` property to the Selected value of the Show Temp Checkbox component.



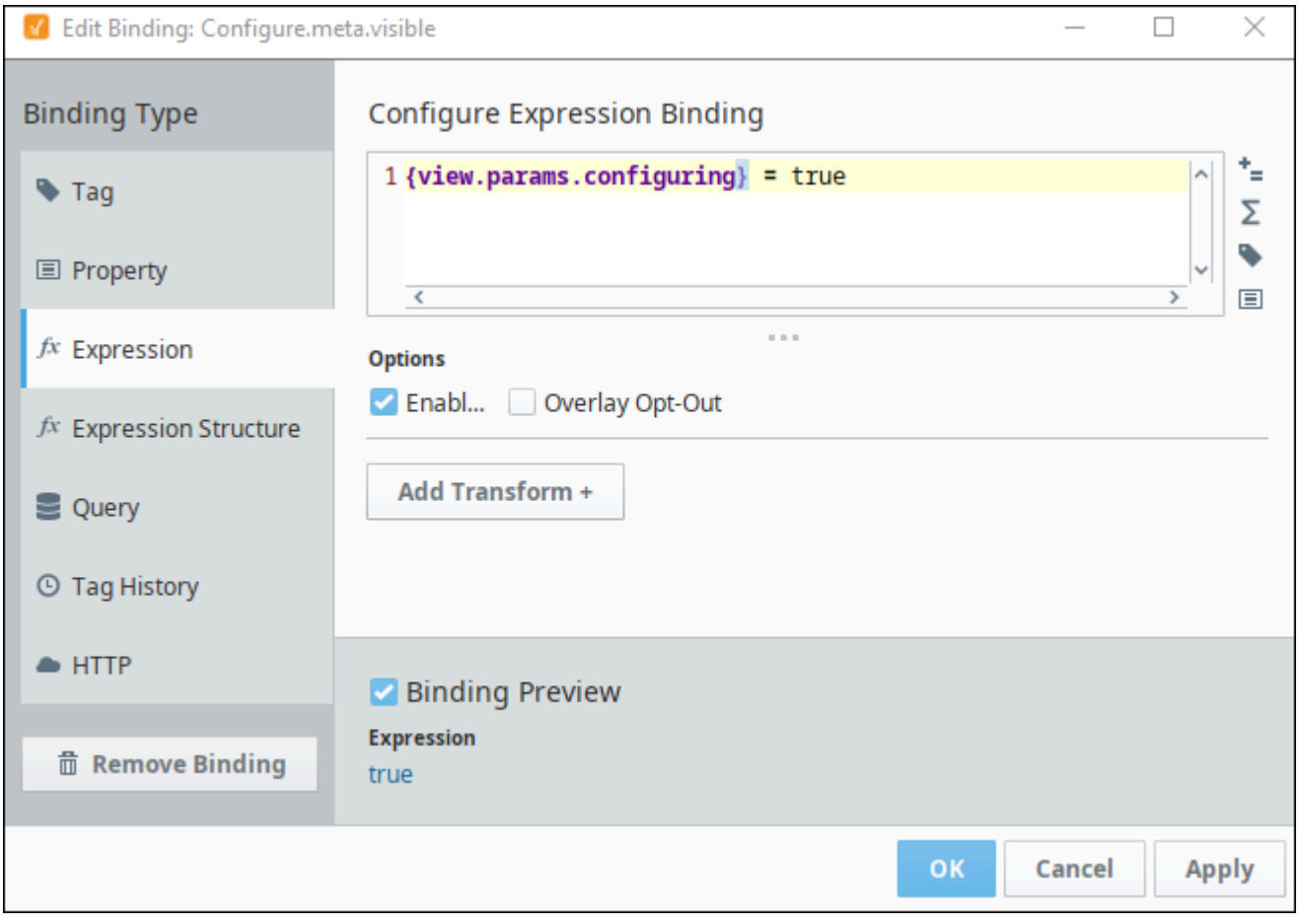
8. Create the following bindings to show one container at at time.

> **NOTE**
>
> If you used a Flex container at the start of this example, use the `display` property instead of the `visible` property for the following steps.

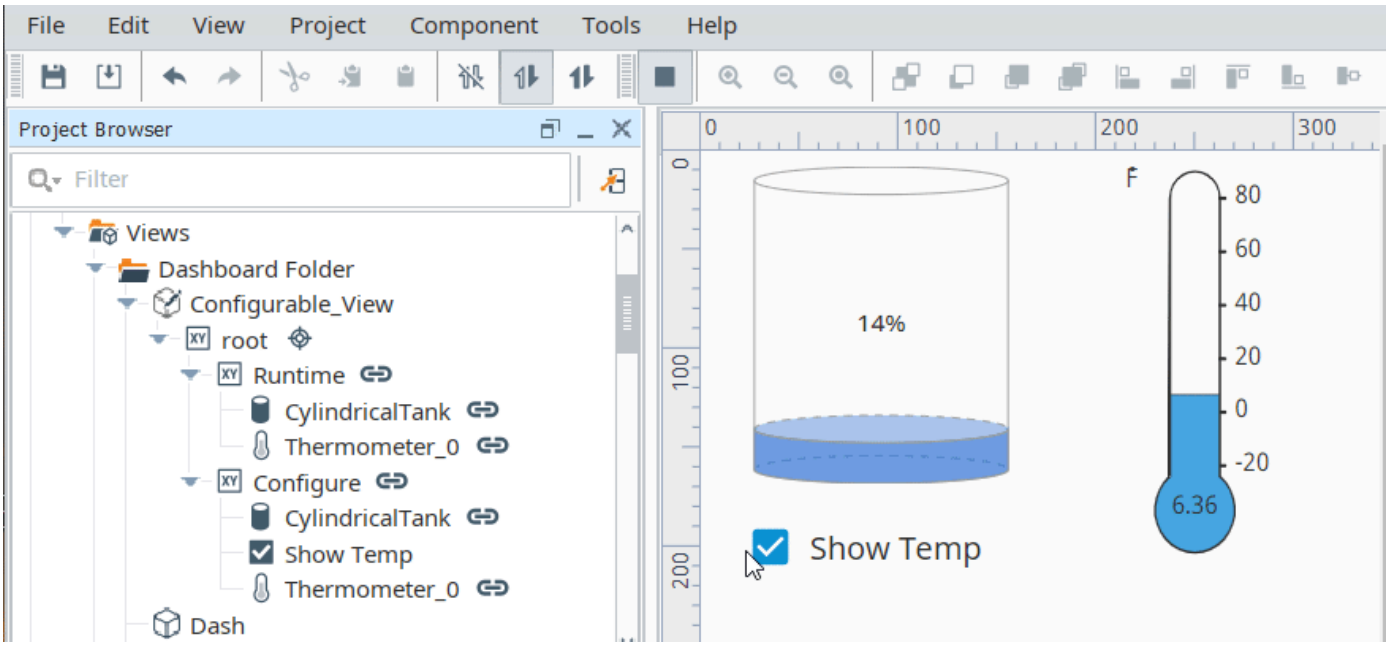
i. Bind the `visible` property for the Runtime container using an expression. It should be true when the configuring param is false.



ii. Bind the `visible` property for the Configure container using an expression. It should be true when the configuring param is true.



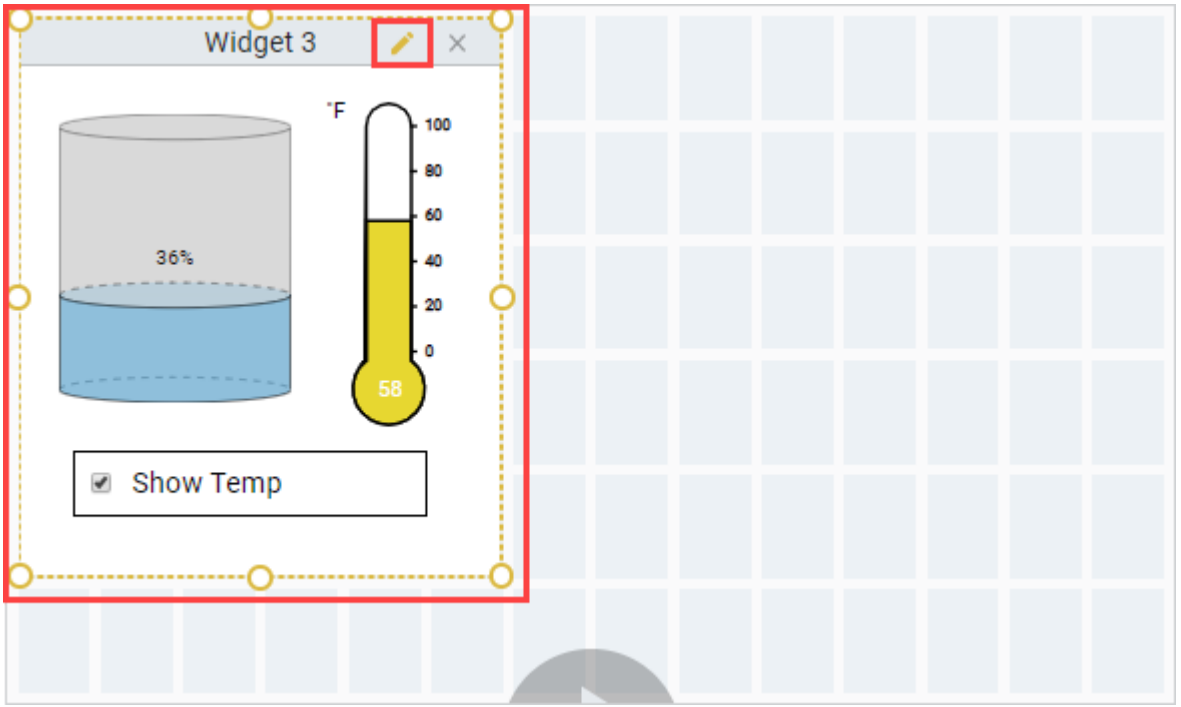
9. Save your project, and then put the Designer into Preview mode. When you check the Temp Show box, you'll see the Thermometer component appear.



Now you can use this view in your dashboard.

Configuring the Dashboard Component in a Perspective Session

To use a configurable widget, like the one set up above, in the runtime, make sure the `editingToggle` property is checked in the Designer. Now in the Session, users can click the Edit icon on the component. This will change the view to show the grid system and allow users to configure the widgets.



Saving Perspective Session Edits and Populating Widgets

Edits that an end user makes in their dashboard in a Perspective Session are not automatically saved and do not persist when the end user's Session restarts. A Session can be refreshed within the same Session. One possible solution for populating widgets for the next editing session is to add a property change script on the `widgets` prop to listen for changes and then write that value back to a database along with any user information derived from the active Session. The value of the `widgets` prop will be an array of QualifiedValues, which you'll need to handle accordingly. In similar fashion, consider adding an `onStartup` event action that will query the database and then populate the `widgets` prop with the users last saved configuration and optionally populate the `availableWidgets` prop (possibly for varying user roles).

Saving and Loading Dashboard Component JSON Data

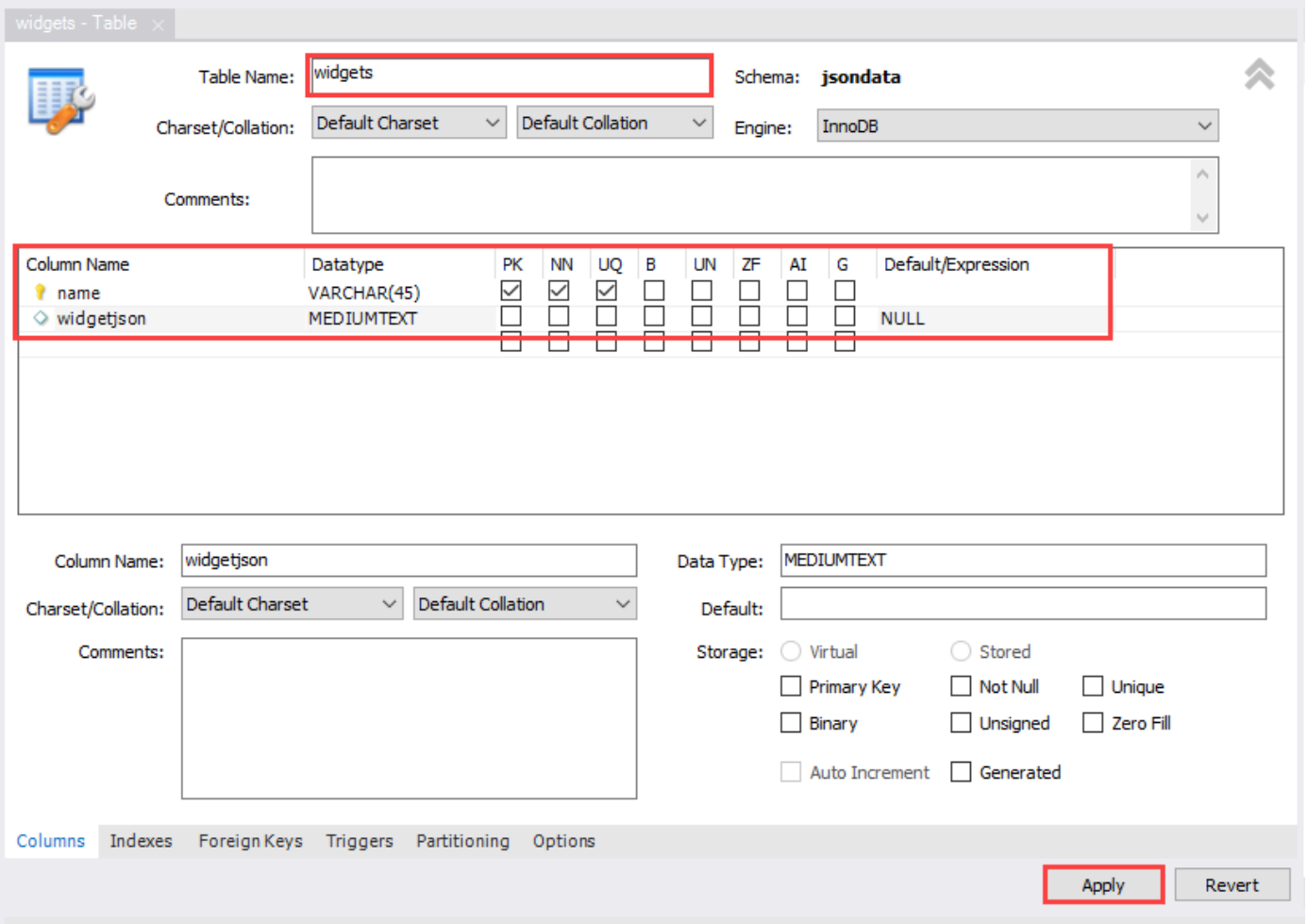
The Dashboard component also has the ability to save and load custom widget configurations. This functionality can be set up by configuring a database table to store widget data, and then writing scripts to save and pull JSON data. We'll set up an example using a MySQL database to demonstrate the basic format of this functionality, which can be expanded to serve many processes.

Once you have a database connection, complete the following steps to create a MySQL database table:

1. Open MySQL Workbench and create a table.
2. Enter a name for the table, in this case we used widgets.
3. Enter name under Column Name in the first row.
4. Use the dropdown to select VARCHAR(45) from the Datatype dropdown.
5. Select UQ to make sure values aren't duplicated.
6. Enter widgetjson under Column Name a second row.
7. Select MEDIUMTEXT from the Datatype dropdown.

> **NOTE**

> Depending on your database, it may be a different datatype name so make sure to select a datatype that can hold long strings.



8. Select Apply.
9. Select Apply and Finish on the confirmation pop-up.
10. Add data into your Workbench table query scripting:

```
INSERT INTO test.widgets VALUES ("Widget", "1234")
```

11. Select the Execute  icon to confirm the Action Output.

Next, we will create two named queries in the Designer that will be used in the save and load scripts.

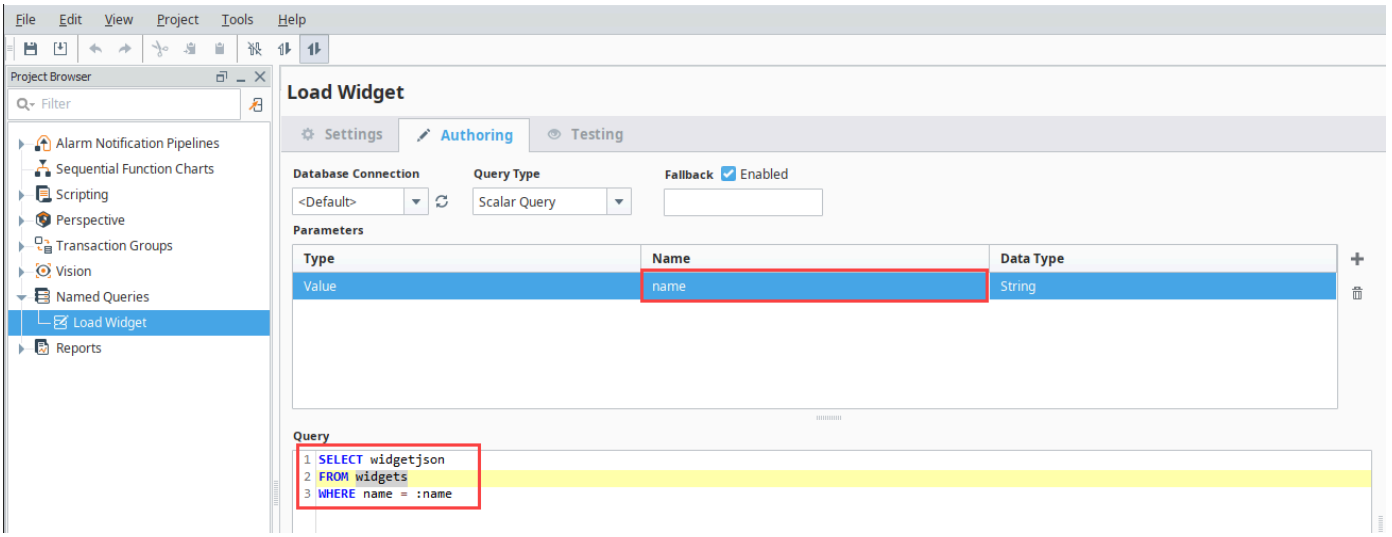
1. Right-click Named Queries and select New Query in the Designer.
2. Enter Load in the Name field.
3. Select Create Named Query.

4. Select the Authoring Tab.

5. Fill in the Query script:

```
SELECT widgetjson
FROM widgets
WHERE name = :name
```

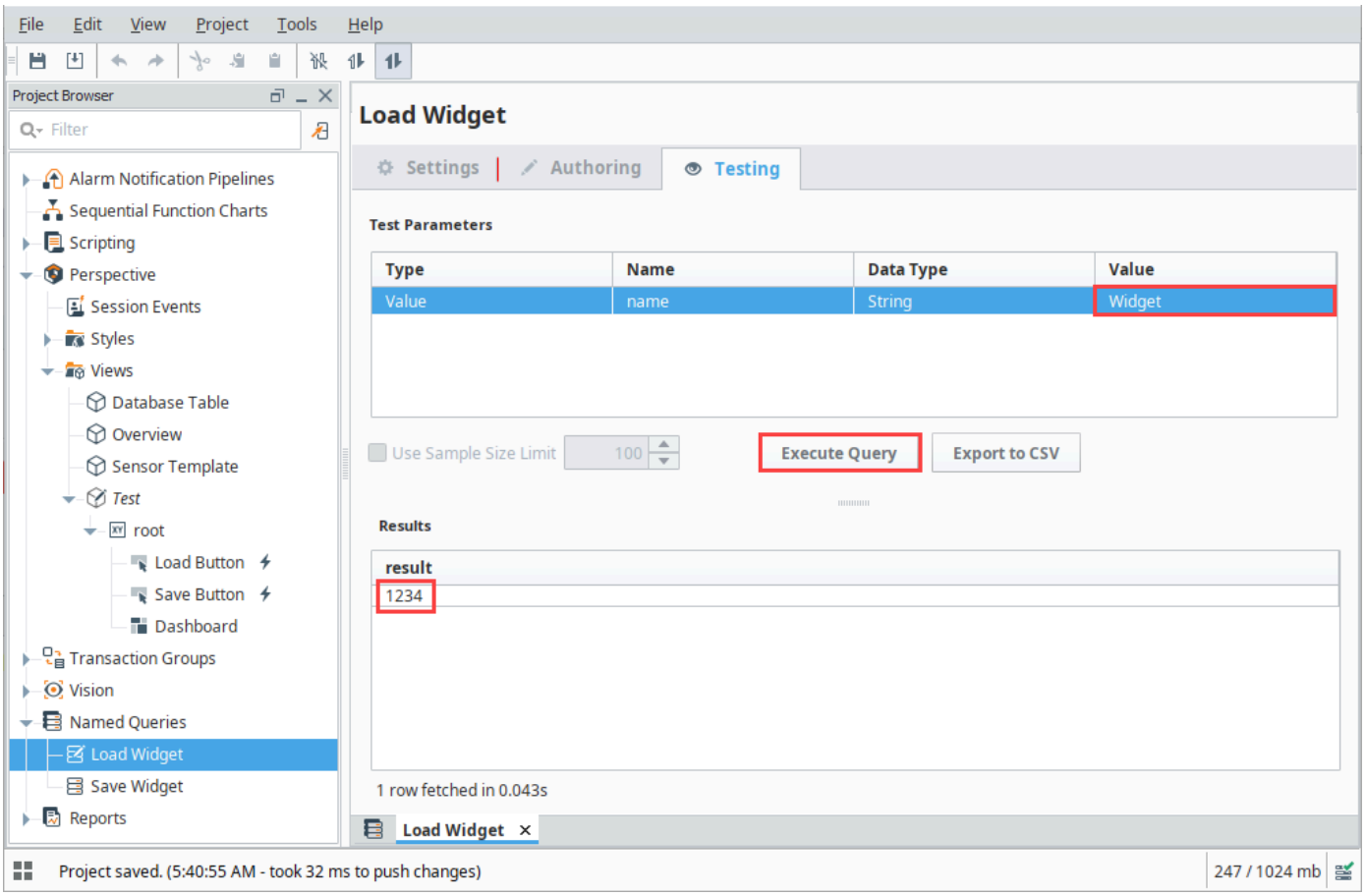
6. Enter name in Name column.



7. Select the Testing Tab.

8. Enter a name value in the Value column, in this example, we used Widget.

9. Select Execute Query to confirm the result is as expected, 1234.



10. Select the Save icon to save the Load Widget Query.

11. Right-click Named Queries and select New Query.

12. Enter Save in the Name field.

13. Select Create Named Query.

14. Select the Authoring Tab.

15. Fill in the Query script:

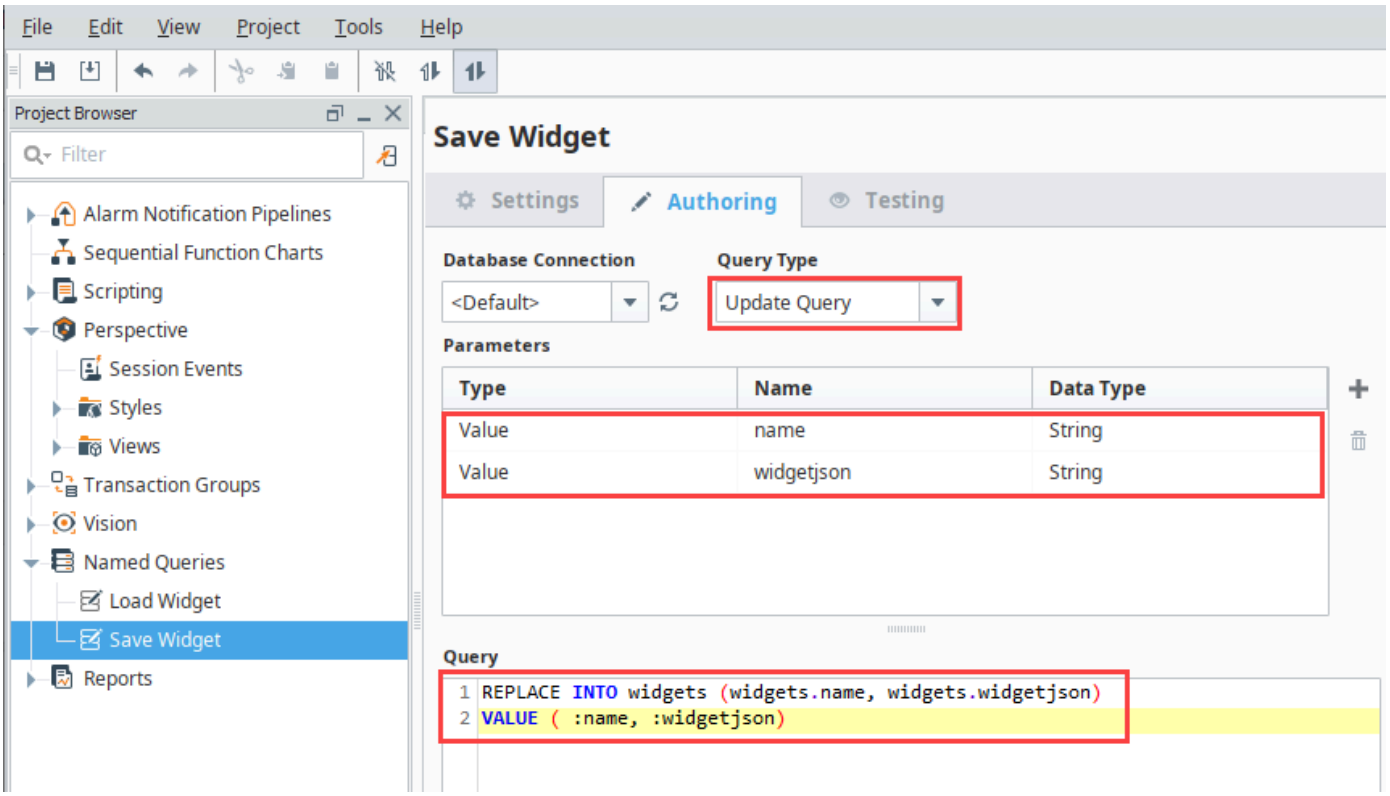
```
REPLACE INTO widgets (widgets.name, widgets.widgetjson)
VALUE ( :name, :widgetjson)
```

16. Select Update Query as the Query Type.

17. Enter name into the Name field for the first parameter row.

18. Enter widgetjson into the Name field for the second parameter row.

19. Confirm both row Data Types are listed as String.



20. Select the Testing tab.

21. Enter Widget into the name row.

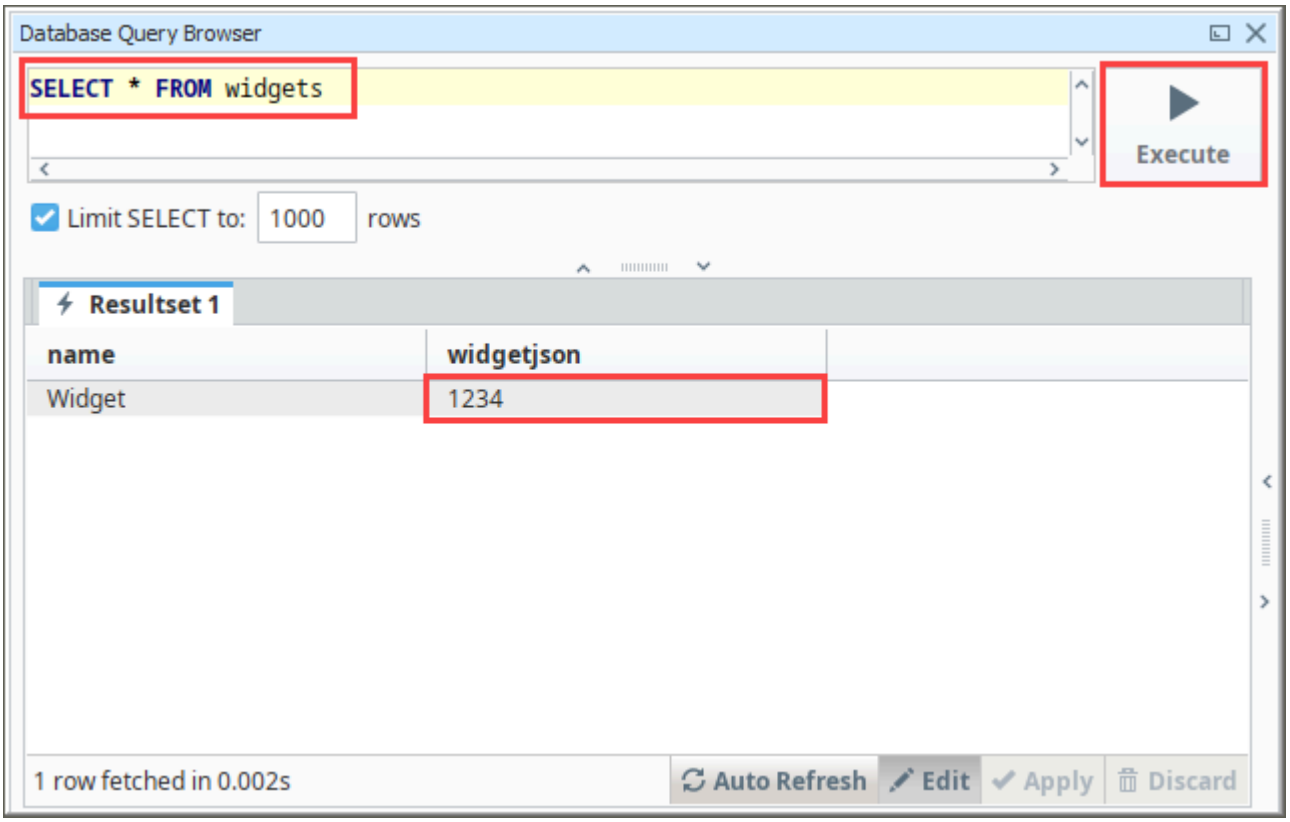
22. Enter 5678 into the widgetjson row.

23. Open the Database Query Browser to see how the saved value updates.

24. Enter browsing script to see the current widgets results:

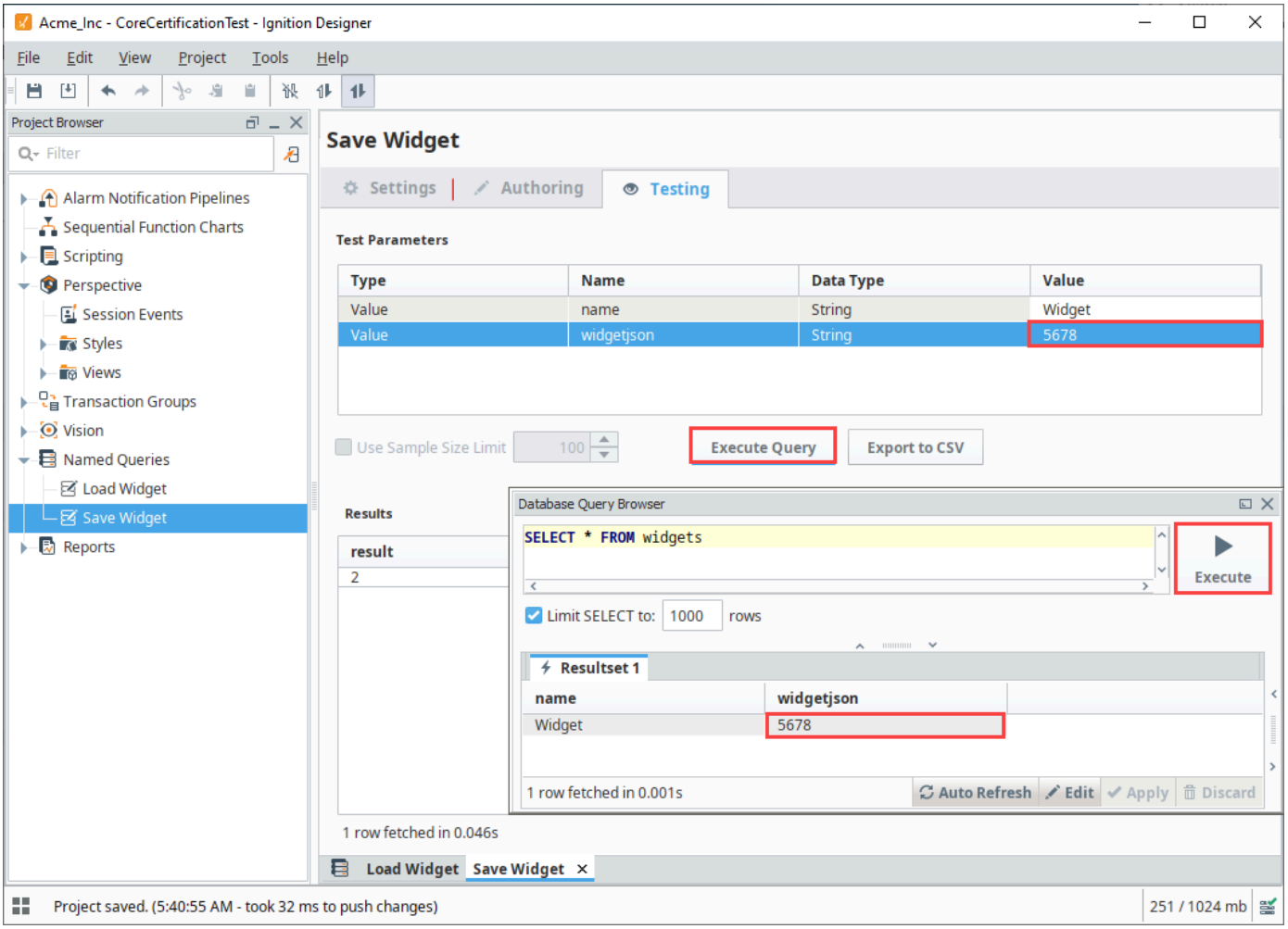
```
SELECT * FROM widgets
```

25. Select Execute to see Widget listed in the name column and 1234 in the widgetjson column.



26. Now, select Execute Query for the Save Widget Testing to save a new widgetjson value.

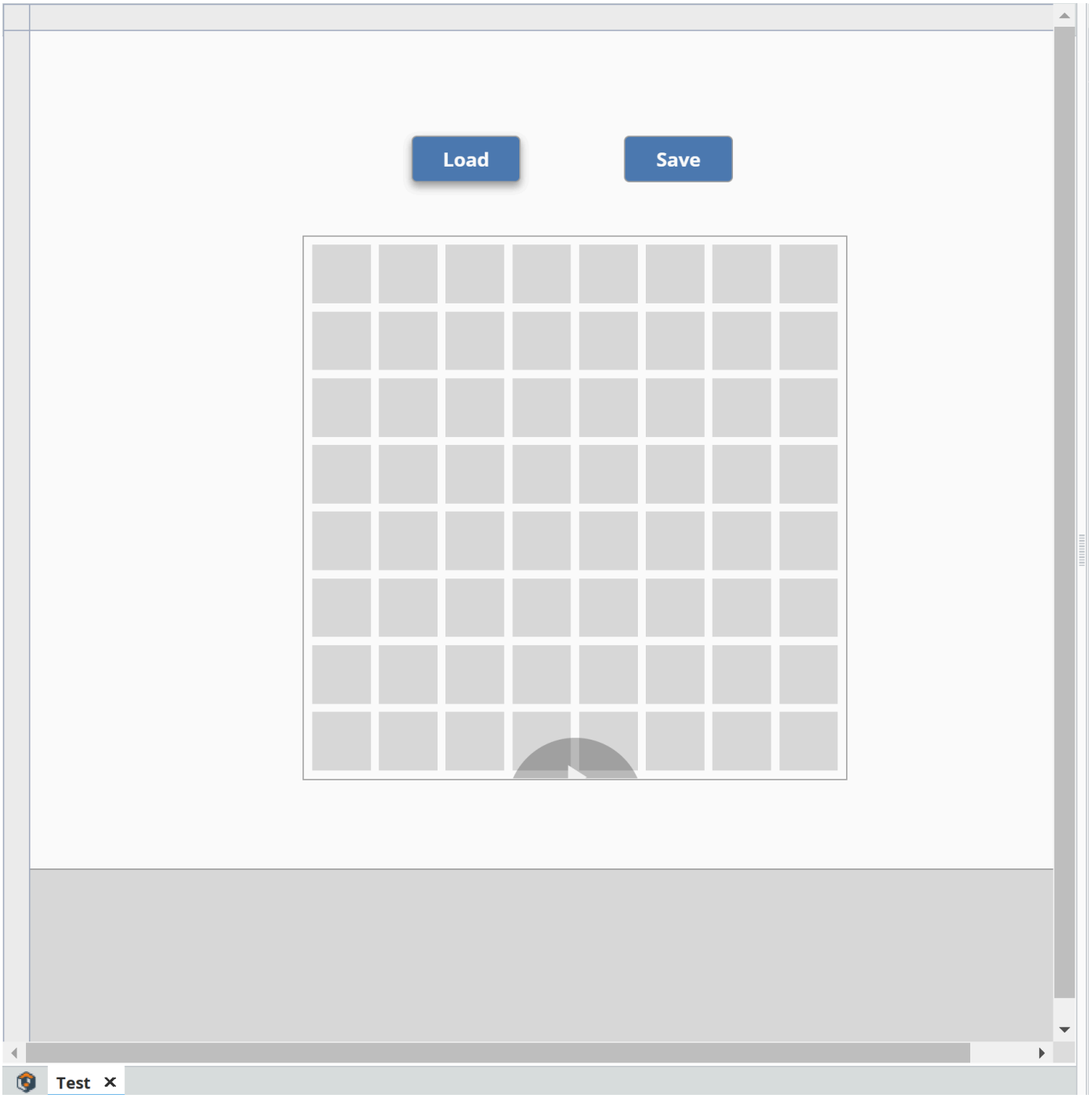
27. Select Execute on the Database Query Browser to see the updated result of 5678 in the widgetjson column.



28. Select the Save icon to save the Save Widget Query.

Next we will create a New View to enable the save and load functionality.

1. Right-click Views and select New View.
2. Enter a name into the Name field. In this example, we used Test as the name. Root Container Type selection does not matter for this example. Selecting Page URL is optional, but can be helpful. In this example, we use the page URL to see the View when we launch a Session at the end of the example setup.
3. Drag a Dashboard component onto the view, drag two buttons onto the view.
4. Change one button text to Load and the other to Save.

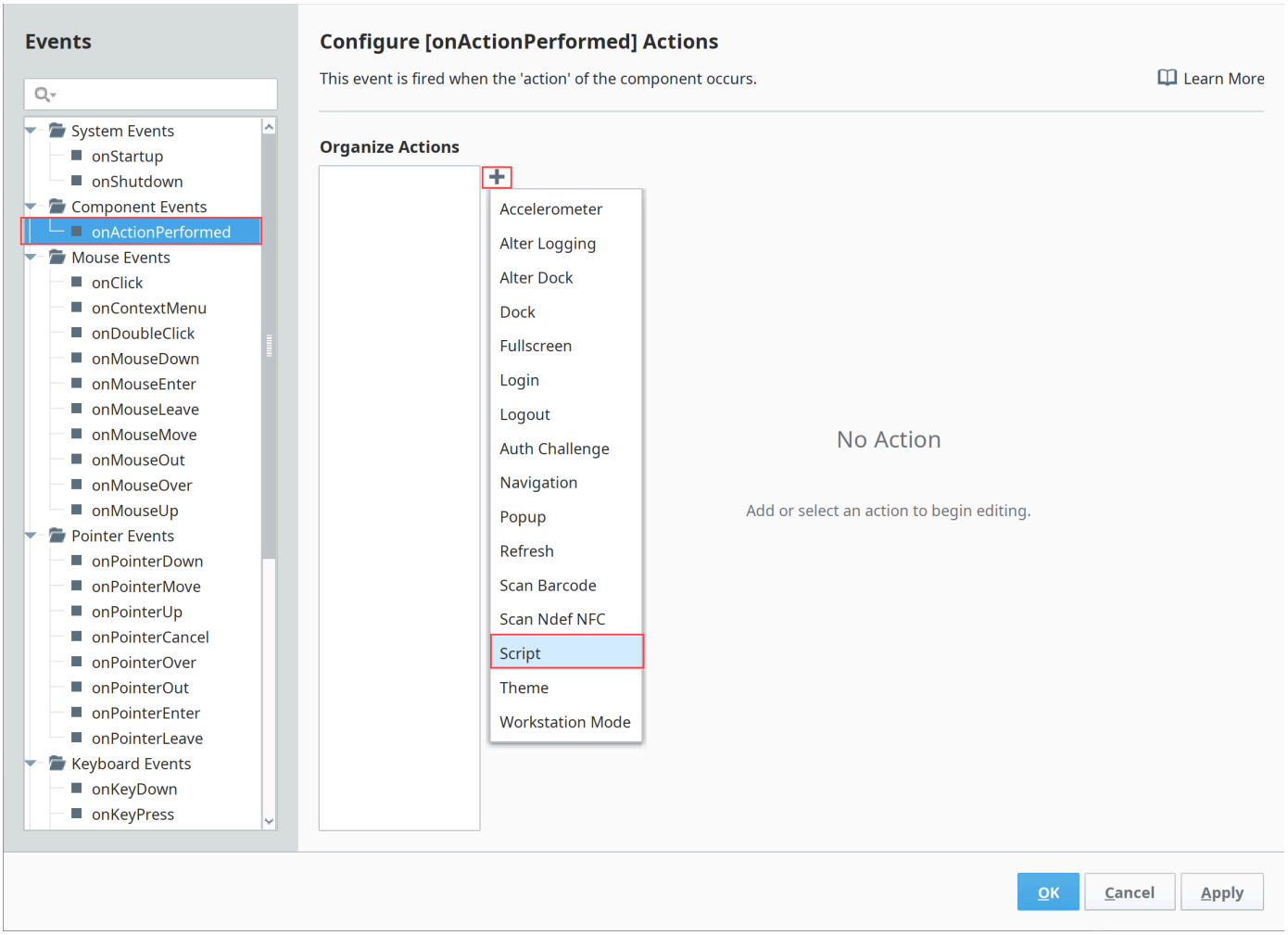


5. Right-click on the Save button and select Configure Events…

6. Select onActionPerformed.

7. Select the Add ➕ icon.

8. Select Script.



9. Enter scripting to pull name and widgets out of properties:

```
name = "Widget 1"
widgetData = self.getSibling("Dashboard").props.widgets
```

> **NOTE**
>
> The name can alternatively come from a text entry box.

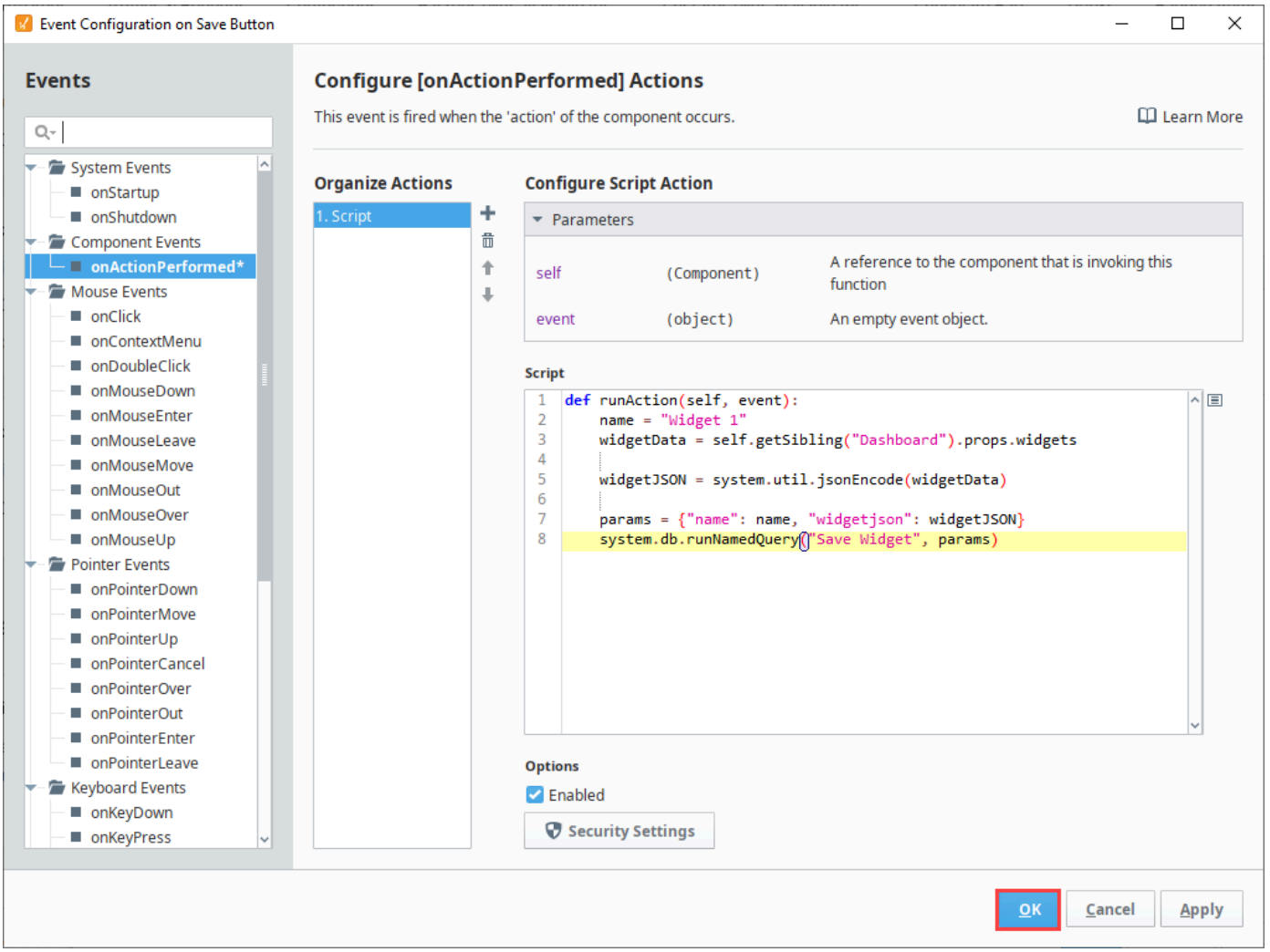
10. Enter scripting to convert the widget json format to a string:

```
widgetJSON = system.util.jsonEncode(widgetData)
```

11. Enter scripting to update variables for the named query and run the query:

```
params = {"name": name, "widgetjson": widgetJSON}
system.db.runNamedQuery("Save Widget", params)
```

12. Select OK.



13. Right-click on the Load button and select Configure Events…

14. Select onActionPerformed.

15. Select the Add ✚ icon.

16. Select Script.

17. Enter scripting to prepare the name of the widgets we want to load:

```
name = "Widget 1"
params = {"name": name}
```

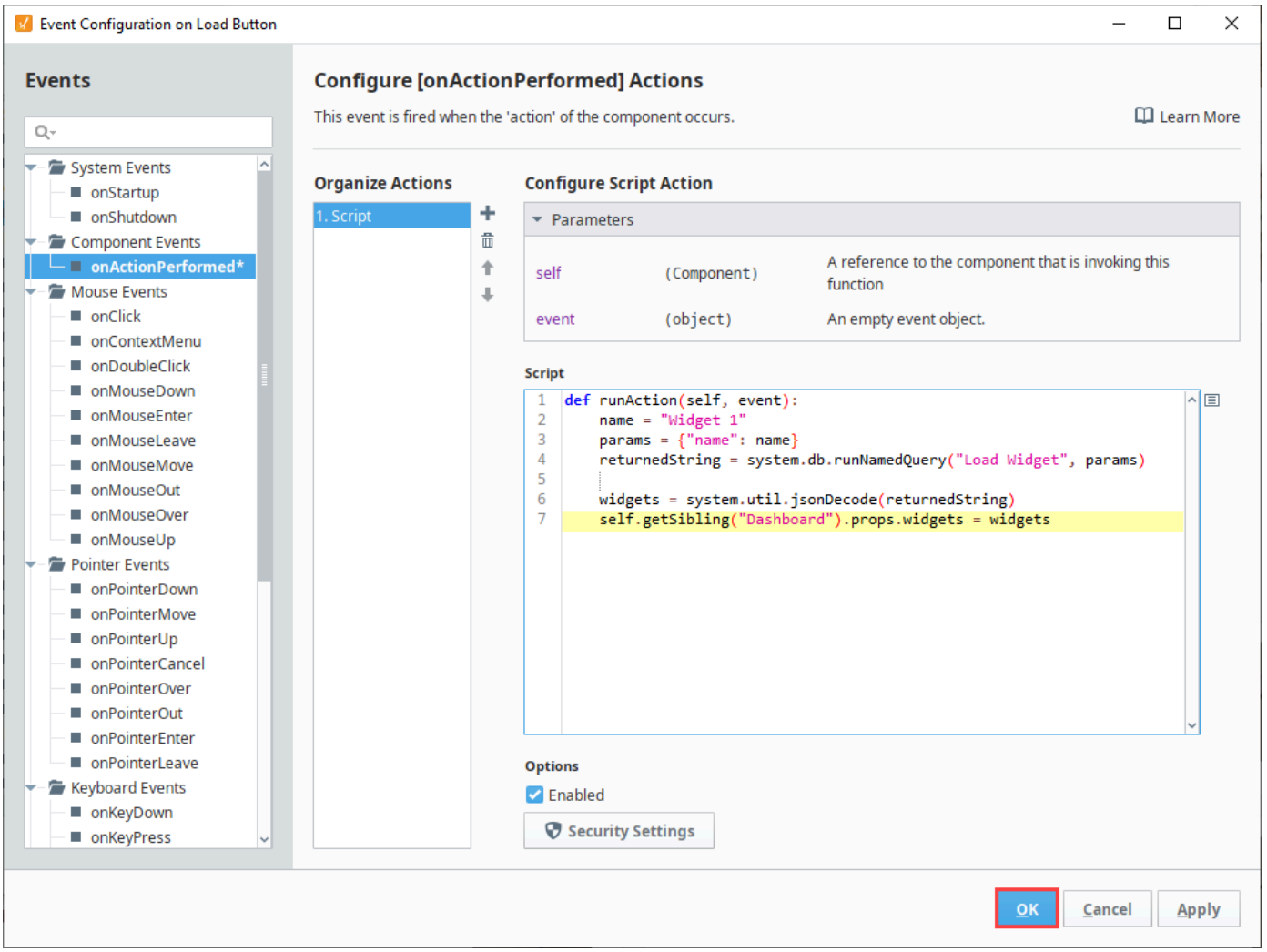
18. Enter scripting to run the Load query:

```
returnedString = system.db.runNamedQuery("Load Widget", params)
```

19. Enter scripting to decode the string into the JSON object and set the Dashboard to the corresponding widgets:

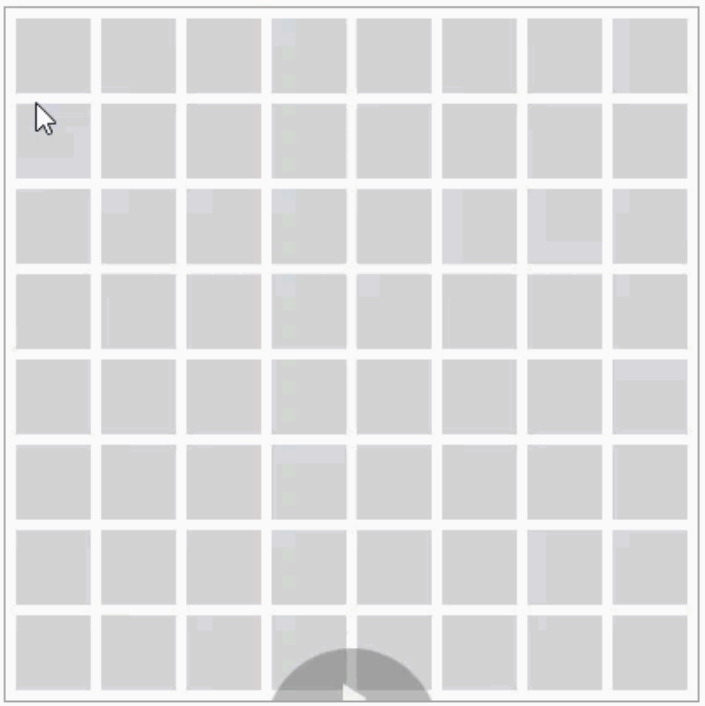
```
widgets = system.util.jsonDecode(returnedString)
self.getSibling("Dashboard").props.widgets = widgets
```

20. Select OK.

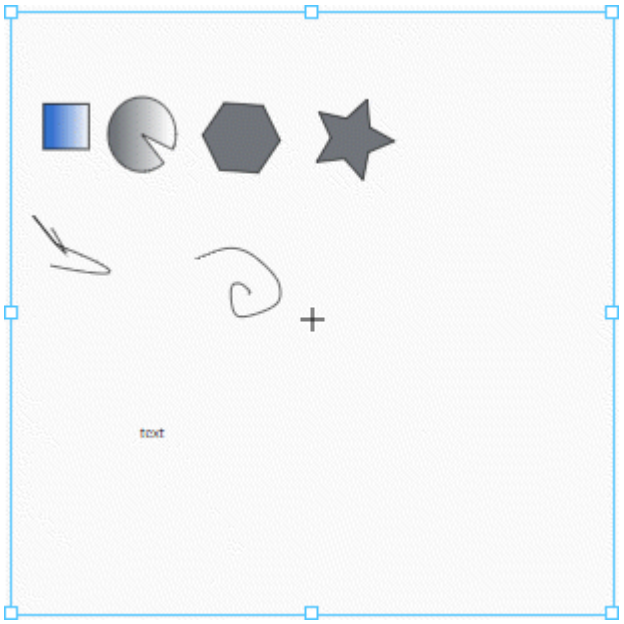


Now you can test your Save and Load button functions using the Preview Mode in the Designer or Launching a Perspective Session. Testing will show how users can add, remove, and modify widgets freely, the save the Dashboard configuration so if they continue to modify widgets, the Load button can be used to return the Dashboard to the saved setup.

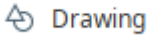
Load Save



Perspective - Drawing



Component Palette Icon:



Description

A component that allows you to create and manipulate SVG-based vector graphics directly within the Designer. You can configure these drawings through the built-in Drawing Editor.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
`viewBox`	Defines the visible area and coordinate system for the drawing. If null, the entire Canvas is visible.	value: object
`preserveAspectRatio`	Controls how the drawing scales relative to the Viewbox, preserving or ignoring the aspect ratio. The default value, `xMidYMid`, centers the drawing while maintaining the aspect ratio.	value: string
`elements`	A dynamic list of SVG elements within the drawing, stored as JSON. Automatically updates when elements are created or modified. See **Element Properties** for details about each element.	value: array
`style`	Sets a style for the entire drawing. The Style menu contains all tools for modifying appearance, including fill, stroke, and opacity. You can also specify a **style class**.	value: object

Elements Properties

The following table displays the attributes within the elements property. With a few exceptions, these attributes are camel-cased versions of the **original** attribute. These exceptions will be denoted with an asterisk (*).

> **CAUTION**
>
> SVG attributes are expansive, and some attributes have been modified to be more compatible with Ignition. Testing these attributes is recommended to avoid unexpected behavior.

Name	Description	Property Type
clipPath	Defines a clipping path with the element the path is related to.	value: object
clipRule	Defines an algorithm that determines the graphical elements contained within a clipPath element.	value: string
elements	Allows users to nest individual elements within the property tree.	value: array
*end	Adds a marker to the end of the shape path. This attribute has been modified to automate some aspects that the original `marker-end` attribute required manual setup or manipulation for, including unique IDs for itself and referenced elements for color, rotation, width, height, and more. It also has built-in shapes such as arrows, reversed arrows, circles, squares, and lines.	value: object
fill	Configuration options for the color inside a shape.	value: string or object
id	A DOM id that will be automatically prepended with prefix to prevent collisions with other DOM ids. Manually assigning new ids will break rendering of elements referencing this id.	value: string
mask	An attribute that allows users to bind a given mask element to the element the attribute belongs to.	value: object
*name	A name to be displayed for the specified element within the Drawing Editor Layer's panel.	value: string
opacity	Sets the transparency of an element and its children.	value: string
overflow	Behavior when an element's context is too big to fit in its block formatting context. Browser support varies.	value: string

Name	Description	Property Type
pathLength	Advanced SVG attribute representing the total length for the path, in user units.	value: numeric
*start	Adds a marker to the start of the shape path. This attribute has been modified to automate some aspects that the original `marker-start` attribute required manual setup or manipulation for, including unique IDs for itself and referenced elements for color, rotation, width, height, and more. It also has built-in shapes such as arrows, reversed arrows, circles, squares, and lines.	value: object
stroke	Configuration options for the color of an outline of a shape.	value: string or object
style	Allows users to style an element using CSS declarations.	value: object
*textPath	Renders text along the shape of a path. Applicable to text type elements only.	value: object
transform	A string representation of SVG transform definitions that are applied to an element and the element's children.	value: string
*type	Controls the type of element being rendered.	value: string
*typeAttr	**New in 8.3.4** The type attribute to be applied to the SVG element.	value: string
vectorEffect	Advanced SVG attribute specifying an effect to be used when drawing an SVG element.	value: string
*vertex	Adds a marker to the interior vertices of the shape path, excluding the first and last vertices. This attribute has been modified to automate some aspects that the original `marker-mid` attribute required manual setup or manipulation for, including unique IDs for itself and referenced elements for	value: object

Name	Description	Property Type
	color, rotation, width, height, and more. It also has built-in shapes such as arrows, reversed arrows, circles, squares, and lines.	
visibility	Allows users to control the visibility of graphical elements.	value: string
*wrapper	If provided, this element will be wrapped in a wrapper SVG element of this wrapperType. This is useful for clipPath, mask, and pattern.	value: object

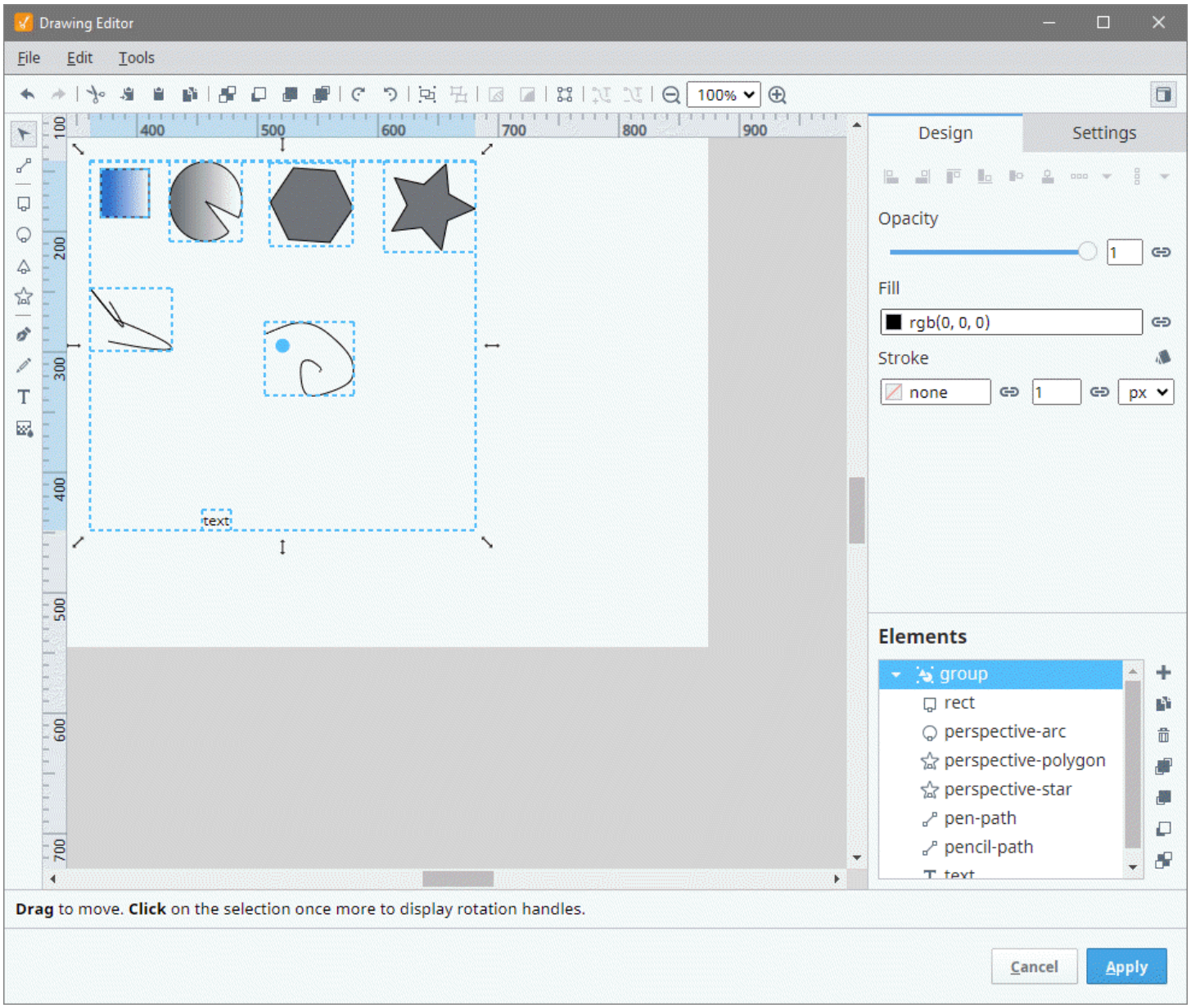
Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Drawing Editor

The Drawing Editor can be opened in the Designer by right-clicking on the Drawing component within a Perspective view or from the Project Browser, and selecting **Edit Drawing**. You can open only one drawing at a time in the editor. While a drawing is being edited, the right-click context menu option on drawing instances in the Designer will be disabled.

The Drawing Editor supports importing vector graphics (SVGs) using drag and drop functionality.



The **Drawing Editor** interface consists of several sections to facilitate creating and editing drawings:

- **Command Bar**: Provides quick access to core actions like applying changes, undo/redo, and managing objects.
- **Toolbox**: Contains all the drawing tools used to create and modify shapes, lines, and text.
- **Design Panel**: Provides alignment, properties for selected elements, and group management options.
- **Settings Panel**: Allows you to configure the Canvas, grids, guides, and other workspace properties.
- **Viewbox**: Displays the current viewable area of the Canvas, defining the visible portion of the drawing.
- **Canvas**: The central drawing area where you create and edit vector graphics.
- **Status Bar**: Shows useful information like cursor position, zoom level, and editing tips based on the current tool or action.

- **Apply / Cancel Buttons**: Located at the bottom of the editor, these buttons allow you to either save your changes or discard them.

Command Bar



The **Command Bar** gives you easy access to essential drawing actions, including applying changes, undoing or redoing edits, and managing object layers or alignment. Depending on the tool selected from the Toolbox, additional tool-specific controls may populate in the **Command Bar**.

Icon	Command	Description
	**Undo**	Reverts the most recent change.
	**Redo**	Reapplies the most recent undone change.
	**Cut**	Removes the selected object and places it on the clipboard.
	**Copy to Clipboard**	Copies the selected object to the clipboard.
	**Paste**	Pastes objects from the clipboard into the Canvas.
	**Duplicate**	Creates a duplicate of the selected object.
	**Send to Back**	Moves the selected object to the back of the stacking order.
	**Send Backward**	Moves the selected object one step backward in the stacking order.
	**Bring Forward**	Moves the selected object one step forward in the stacking order.
	**Bring to Front**	Moves the selected object to the front of the stacking order.
	**Rotate Right**	Rotates the selected object 90 degrees to the right.
	**Rotate Left**	Rotates the selected object 90 degrees to the left.
	**Group**	Groups selected objects for unified manipulation.
	**Ungroup**	Separates a grouped object into individual elements.
	**Mask**	Creates a mask using selected objects to hide portions of other objects.

Icon	Command	Description
	**Clip**	Creates a clipping path using selected objects.
	**Convert to Path**	Converts selected objects into an editable path object.
	**Place Text on Path**	Places selected text along a path.
	**Remove Text from Path**	Removes text from a path and reverts it to a regular text object.
	**Zoom Out**	Zooms out, reducing the size of objects on the Canvas.
	**Zoom In**	Zooms in, increasing the size of objects on the Canvas.
	**Show / Hide Panel**	Toggles the visibility of the side panel.

For detailed descriptions of the additional tool controls that appear in the Command Bar, see the **Path Tool Controls**, **Pen Tool Controls**, and **Pencil Tool Controls** sections.

Keyboard Shortcuts

New in 8.3.1

You can use keyboard shortcuts in the Drawing Editor to quickly access many of the commands listed above, switch between tools, modify shapes, and perform common editing actions without leaving the Canvas. Shortcut tooltips appear on action buttons, and the Status Bar displays context-aware hints based on the active tool or selection.

Windows/Linux macOS

Tool Activation

Tool	Shortcut
Path	`F1`
Selection	`F2`
Rectangle	`F3`
Circle / Ellipse	`F4`
Polygon	`F5`
Star	`F6`
Pen	`F7`
Pencil	`F8`
Text	`F9`
Gradient	`F10`

Path and Pen

Action	Shortcut
Convert to curved segment	`C`
Convert to straight segment	`1`
Break segment at nodes	`Enter`
Add node between selected nodes	`Insert`
Remove selected nodes	`Delete`
Commit path and close	`Enter`
Cancel current path	`Esc`
Pen: Bézier mode	`B`
Pen: Paraxial mode	`L`
Snap to 15 degrees while drawing	`Ctrl` (hold)
Override paraxial (free move)	`Shift` (hold)

Selection (Move and Scale)

Action	Shortcut
Toggle rotation mode	`R`
Nudge by 1 pixel	Arrow keys
Nudge by 10 pixels	`Alt + Arrow`
Change width by 1 pixel	`Shift + Left/Right`
Change height by 1 pixel	`Shift + Up/Down`
Change width by 10 pixels	`Ctrl + Shift + Left/Right`
Change height by 10 pixels	`Ctrl + Shift + Up/Down`

Common Actions

Action	Shortcut
Cut	`Ctrl+X`
Copy	`Ctrl+C`
Paste	`Ctrl+V`
Duplicate	`Ctrl+D`
Group	`Ctrl+G`
Ungroup	`Ctrl+Shift+G`
Undo	`Ctrl+Z`
Redo	`Ctrl+Y`
Delete	`Delete`
Reload Drawing Editor	`Ctrl+F12`

View

Action	Shortcut
Zoom In	`Ctrl + +` or Mouse Wheel Up
Zoom Out	`Ctrl + -` or Mouse Wheel Down
Fit to Viewbox	`Ctrl+0`
Pan	`Ctrl + Mouse Drag`
Toggle Side Panel	`F9`

Shape Modifiers

Tool	Modifier	Result
Circle Tool	`Ctrl` (hold)	Constrain ellipse to a circle
Rectangle Tool	`Ctrl` (hold)	Constrain rectangle to a square

Toolbox



The **Toolbox** contains all the drawing tools you will use to create and modify shapes, paths, and text within the Canvas. Each tool provides different options depending on what you want to create or edit. Some tools have additional controls available in the **Command Bar** when they are selected.

Icon	Tool	Description
	**Selection**	Selects objects on the Canvas for moving, resizing, or transforming.
	**Path**	Draws and edits complex curves and lines with path options. See **Path Tool Controls** for additional options.
	**Rectangle**	Draws rectangles and squares on the canvas.
	**Circle**	Draws circles and ellipses.
	**Polygon**	Draws polygons with a specific number of corners.
	**Star**	Creates star shapes with adjustable points.
	**Pen**	Allows precise control for creating straight lines or Bézier curves. See **Pen Tool Controls** for additional options.
	**Pencil**	Draws freehand lines with a configurable smoothing factor. See **Pencil Tool Controls** for additional options.
	**Text**	Adds text to the Canvas, which can then be styled, aligned, and manipulated.
	**Gradient**	Applies and edits gradient fills.

Path Tool Controls



The **Path Tool** provides additional controls to manipulate nodes and segments on the path:

Icon	Control	Description
	**Add Node**	Adds a new node to the path.
	**Remove Node**	Removes the selected node from the path.
	**Join Nodes**	Connects two selected nodes to create a segment.
	**Break Nodes**	Splits the segment between two selected nodes.
	**Straighten Segment**	Converts the selected segment into a straight line.
	**Curve Segment**	Converts the selected segment into a curve.
	**Sharpen Nodes**	Makes selected nodes sharp corners.
	**Smooth Nodes**	Makes selected nodes smooth, adjusting their curvature.

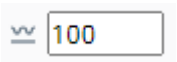
Pen Tool Controls



The **Pen Tool** allows precise control over drawing straight lines and curves:

Icon	Control	Description
	**Bézier Curve**	Draws a Bézier curve segment for greater control over curvature.
	**Paraxial Line Segment**	Draws straight lines constrained to paraxial angles (0°, 90°, etc.).

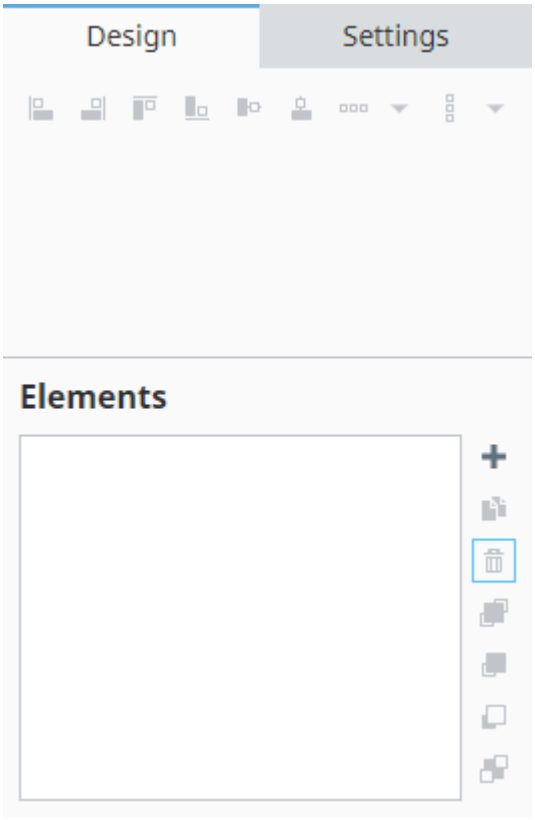
Pencil Tool Controls



The **Pencil Tool** is used for freehand drawing and has a configurable smoothing option:

Icon	Control	Description
	Smoothing	Adjusts the smoothness of the freehand line, reducing jagged edges.

Design Panel



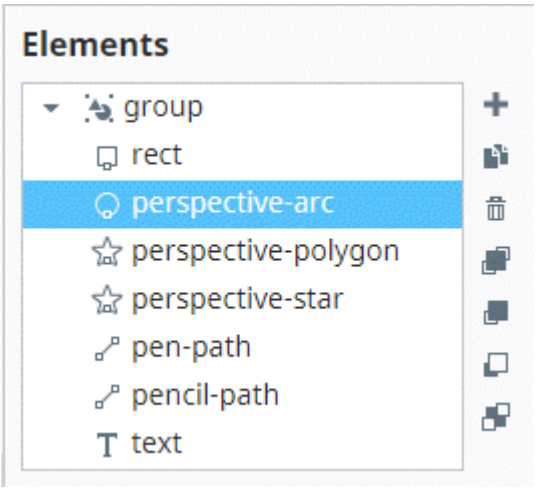
The **Design Panel** is located on the right side of the Drawing Editor interface and contains alignment options, properties for selected elements, and a list of all elements in layer order. This panel is used to control the arrangement and appearance of selected objects, as well as manage the hierarchy of elements in the drawing.

Alignment Options

The alignmentoOptions in the Design Panel help you position and distribute selected objects on the Canvas.

Icon	Action	Description
	Align Left	Aligns selected objects to the leftmost edge.
	Align Right	Aligns selected objects to the rightmost edge.
	Align Top	Aligns selected objects to the topmost edge.
	Align Bottom	Aligns selected objects to the bottommost edge.
	Center Horizontally	Centers selected objects horizontally.
	Center Vertically	Centers selected objects vertically.
	Align as Column	Arranges selected objects in a vertical column.
	Align as Row	Arranges selected objects in a horizontal row.

Elements



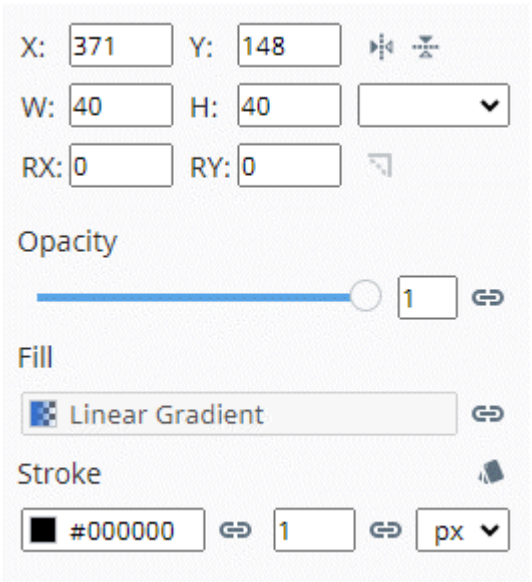
At the bottom of the Design Panel, the Elements subsection displays all elements in the drawing, listed in their layer order. You can manage the elements using the following icons located to the right of the list:

Icon	Action	Description
	**Add Group Element**	Adds a new group to organize elements.
	**Duplicate Element**	Duplicates the selected element.
	**Delete Element**	Deletes the selected element.
	**Bring to Front**	Moves the selected element to the front of the layer stack.
	**Bring Forward**	Moves the selected element forward by one layer.
	**Send Backward**	Moves the selected element backward by one layer.
	**Send to Back**	Moves the selected element to the back of the layer stack.

Element Properties

Each element type has its own properties available in the Design Panel. Below are the tables for each type.

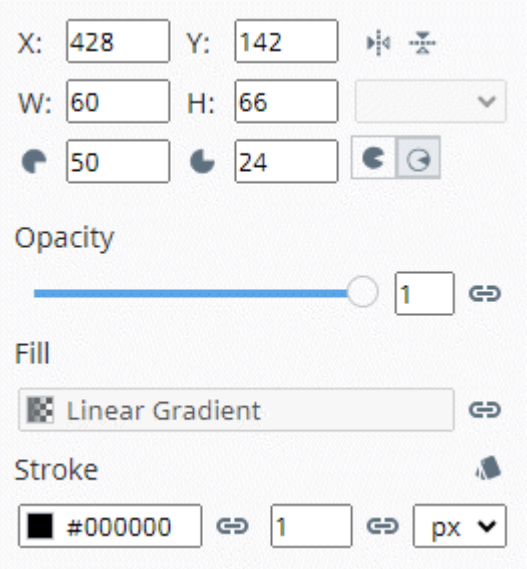
rect Element Properties



Property	Description	Property Type
X	The x-coordinate of the rectangle's top-left corner.	value: numeric
Y	The y-coordinate of the rectangle's top-left corner.	value: numeric
Mirror Horizontally 	Flips the rectangle horizontally.	value: boolean
Mirror Vertically 	Flips the rectangle vertically.	value: boolean
W	The width of the rectangle.	value: numeric
H	The height of the rectangle.	value: numeric
Units	The units for width and height.	value: string
RX	Horizontal corner radius for rounding.	value: numeric
RY	Vertical corner radius for rounding.	value: numeric
Make Corners Straight 	Removes corner rounding, making all corners straight.	value: boolean
Opacity	Sets the rectangle's opacity.	value: numeric
Fill	The fill color of the rectangle.	value: color

Property	Description	Property Type
Stroke	The stroke color of the rectangle. Click the **Configure Stroke** icon for access to additional properties. Refer to the **Stroke Configuration Menu** for a full list of properties.	value: color

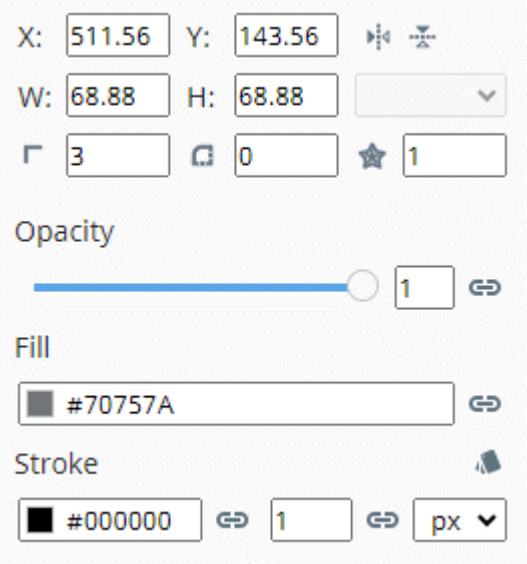
perspective-arc Element Properties



Property	Description	Property Type
X	The x-coordinate of the arc's starting point.	value: numeric
Y	The y-coordinate of the arc's starting point.	value: numeric
Mirror Horizontally 	Flips the arc horizontally.	value: boolean
Mirror Vertically 	Flips the arc vertically.	value: boolean
W	The width of the arc.	value: numeric
H	The height of the arc.	value: numeric
Units	The units for width and height.	value: string
Horizontal Radius 	The radius along the horizontal axis.	value: numeric
Vertical Radius 	The radius along the vertical axis.	value: numeric
Large Arc / Sweep 	Toggle between large arc and sweep direction.	value: toggle
Opacity	Sets the arc's opacity.	value: numeric
Fill	The fill color of the arc.	value: color

Property	Description	Property Type
Stroke	The stroke color of the arc. Click the **Configure Stroke** icon for access to additional properties. Refer to the **Stroke Configuration Menu** for a full list of properties.	value: color

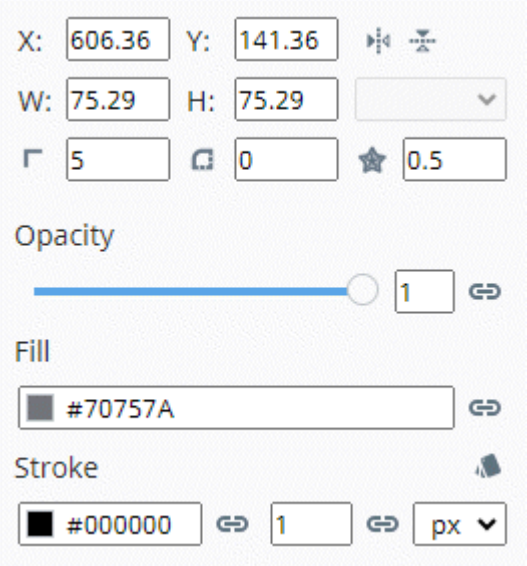
perspective-polygon Element Properties



Property	Description	Property Type
X	The x-coordinate of the polygon’s position.	value: numeric
Y	The y-coordinate of the polygon’s position.	value: numeric
Mirror Horizontally 	Flips the polygon horizontally.	value: boolean
Mirror Vertically 	Flips the polygon vertically.	value: boolean
W	The width of the polygon.	value: numeric
H	The height of the polygon.	value: numeric
Units	The units for width and height.	value: string
Points 	The number of points (vertices) in the polygon.	value: numeric
Corner Radius 	The radius applied to the polygon’s corners.	value: numeric
Base Radius 	The radius from the center to the midpoint of the sides.	value: numeric
Opacity	Sets the polygon's opacity.	value: numeric
Fill	The polygon's fill color.	value: color

Property	Description	Property Type
Stroke	The polygon's stroke color. Click the **Configure Stroke** icon for access to additional properties. Refer to the **Stroke Configuration Menu** for a full list of properties.	value: color

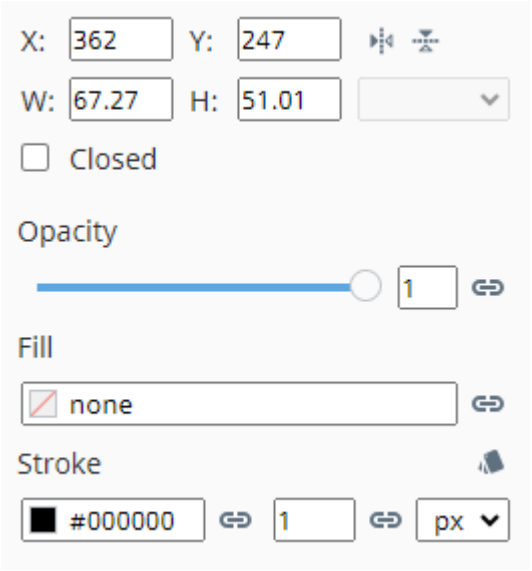
perspective-star Element Properties



Property	Description	Property Type
X	The x-coordinate of the star's position.	value: numeric
Y	The y-coordinate of the star's position.	value: numeric
Mirror Horizontally 	Flips the star horizontally.	value: boolean
Mirror Vertically 	Flips the star vertically.	value: boolean
W	The width of the star.	value: numeric
H	The height of the star.	value: numeric
Units	The units for width and height.	value: string
Points 	The number of points on the star (default: 5).	value: numeric
Corner Radius 	The radius applied to the star's corners.	value: numeric
Base Radius 	The radius from the center to the midpoint of the sides (default: 0.5).	value: numeric
Opacity	Sets the star's opacity.	value: numeric
Fill	The star's fill color.	value: color

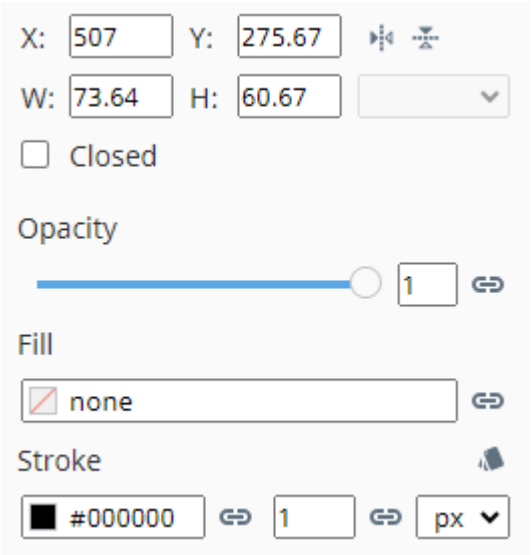
Property	Description	Property Type
Stroke	The star's stroke color. Click the **Configure Stroke** icon for access to additional properties. Refer to the **Stroke Configuration Menu** for a full list of properties.	value: color

pen-path Element Properties



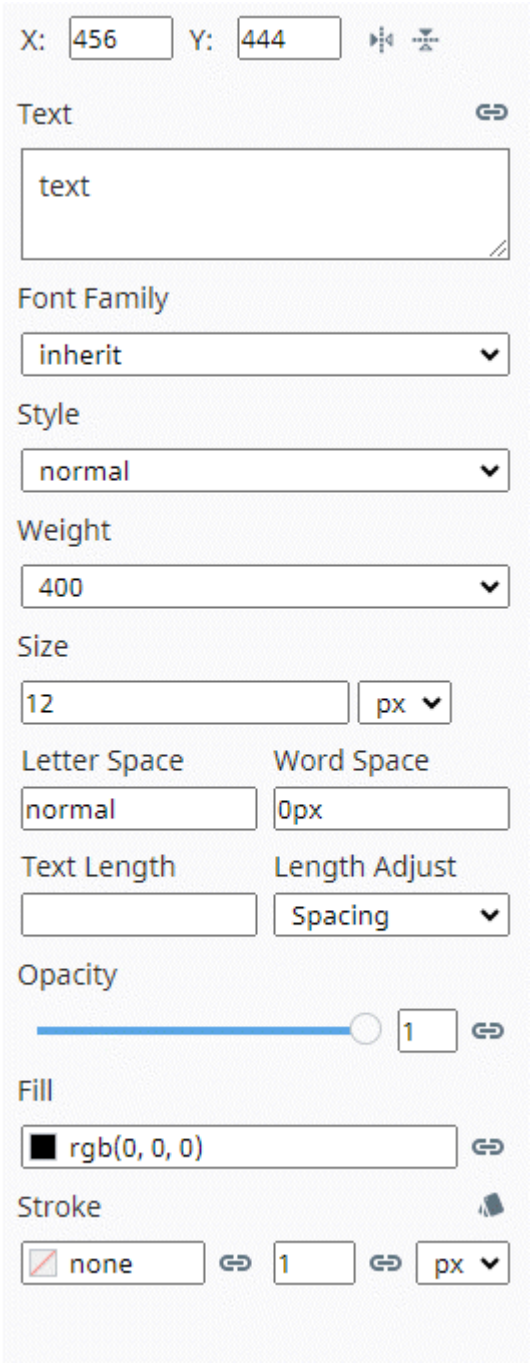
Property	Description	Property Type
X	The x-coordinate of the pen-path's starting point.	value: numeric
Y	The y-coordinate of the pen-path's starting point.	value: numeric
Mirror Horizontally 	Flips the pen-path horizontally.	value: boolean
Mirror Vertically 	Flips the pen-path vertically.	value: boolean
W	The width of the pen-path.	value: numeric
H	The height of the pen-path.	value: numeric
Units	The units for width and height.	value: string
Closed	Closes the path, connecting the last point to the first.	value: boolean
Opacity	Sets the pen-path's opacity.	value: numeric
Fill	The pen-path's fill color.	value: color
Stroke	The pen-path's stroke color. Click the **Configure Stroke** icon  for access to additional properties. Refer to the **Stroke Configuration Menu** for a full list of properties.	value: color

pencil-path Element Properties


X: 507
Y: 275.67
W: 73.64
H: 60.67
Closed
Opacity
1
Fill
none
Stroke
#000000
1
px

Property	Description	Property Type
X	The x-coordinate of the pencil-path's starting point.	value: numeric
Y	The y-coordinate of the pencil-path's starting point.	value: numeric
Mirror Horizontally 	Flips the pencil-path horizontally.	value: boolean
Mirror Vertically 	Flips the pencil-path vertically.	value: boolean
W	The width of the pencil-path.	value: numeric
H	The height of the pencil-path.	value: numeric
Units	The units for width and height.	value: string
Closed	Closes the path, connecting the last point to the first.	value: boolean
Opacity	Sets the pencil-path's opacity.	value: numeric
Fill	The pencil-path's fill color.	value: color
Stroke	The pencil-path's stroke color. Click the **Configure Stroke** icon  for access to additional properties. Refer to the **Stroke Configuration Menu** for a full list of properties.	value: color

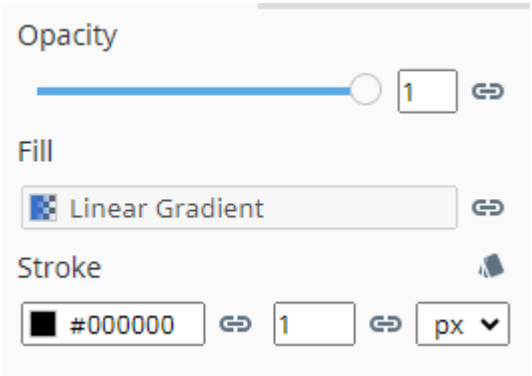
text Element Properties


X: 456
Y: 444
Text
text
Font Family
inherit
Style
normal
Weight
400
Size
12
px
Letter Space
Word Space
normal
0px
Text Length
Length Adjust
Spacing
Opacity
1
Fill
rgb(0, 0, 0)
Stroke
none
1
px

Property	Description	Property Type
X	The x-coordinate of the text's position.	value: numeric
Y	The y-coordinate of the text's position.	value: numeric
Mirror Horizontally 	Flips the text horizontally.	value: boolean
Mirror Vertically 	Flips the text vertically.	value: boolean
Text	The content of the text element.	value: string
Font Family	Specifies the font family (e.g., Arial).	value: string
Style	Sets the font style (normal or italic).	value: string
Weight	Specifies the font weight (normal, bold, etc.).	value: string
Size	Specifies the font size and the units.	value: numeric
Letter Space	Controls the spacing between letters. Default is normal, which applies the default letter spacing of the current font. A value of 0 applies no additional spacing.	value: numeric
Word Space	Controls the spacing between words.	value: numeric
Text Length	Specifies the total length of the text.	value: numeric

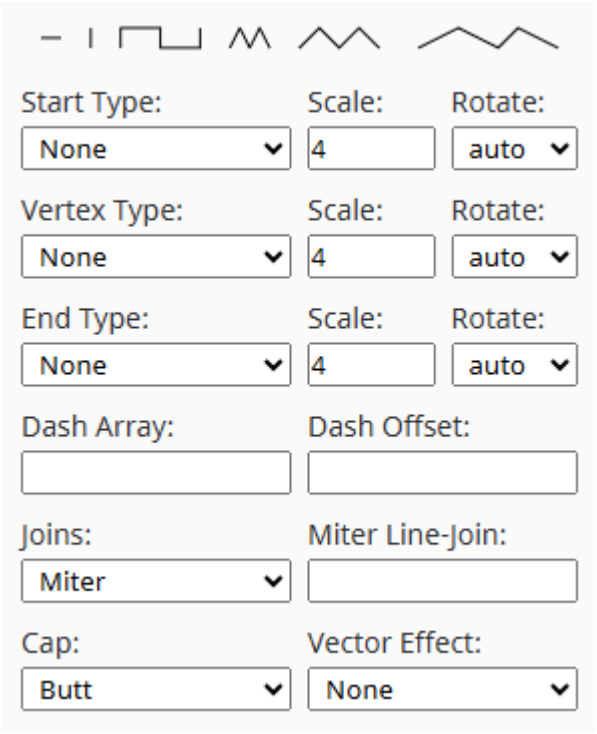
Property	Description	Property Type
Length Adjust	Adjusts the text length to fit the available space.	value: string
Opacity	Sets the text's opacity.	value: numeric
Fill	The text's fill color.	value: color
Stroke	The text's stroke color. Click the **Configure Stroke** icon for access to additional properties. Refer to the **Stroke Configuration Menu** for a full list of properties.	value: color

gradient Element Properties



Property	Description	Property Type
Opacity	Sets the gradient's opacity.	value: numeric
Fill	Applies the gradient to the object's fill.	value: gradient
Stroke	Applies the gradient to the object's stroke. Click the **Configure Stroke** icon for access to additional properties. Refer to the **Stroke Configuration Menu** for a full list of properties.	value: gradient

Stroke Configuration Menu

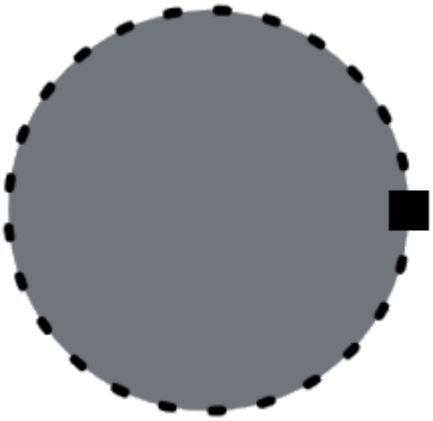


Property	Description
Start Type	Defines the style of the stroke's starting point. • **Scale**: Adjusts the scaling of the stroke at the starting point. • **Rotate**: Rotates the stroke at the starting point.
Vertex Type	Determines how stroke joins are rendered at vertices. • **Scale**: Adjusts the scaling of the stroke at vertices. • **Rotate**: Rotates the stroke at vertices.
End Type	Defines the style of the stroke’s ending point. • **Scale**: Adjusts the scaling of the stroke at the ending point. • **Rotate**: Rotates the stroke at the ending point.
Dash Array	Defines a pattern of dashes and gaps for the stroke, where entered values are read in an alternating pattern of dash and gap lengths. Accepts a comma or spaces to separate values. For example, `6,8` would display a dash of 6 units followed by a gap of 8 units, and so on. If only a single value is provided, it is applied to both the dash and gap length.
Dash Offset	Specifies where to start the dash array pattern along the stroke. Accepts positive and negative values.
Joins	Controls how two segments of a stroke join at a vertex.
Miter Line-Join	Sets the maximum miter length when using miter joins.
Cap	Defines the shape of the stroke's end caps (e.g., butt, round, square).
Vector Effect	Specifies how the stroke is affected by transformations.

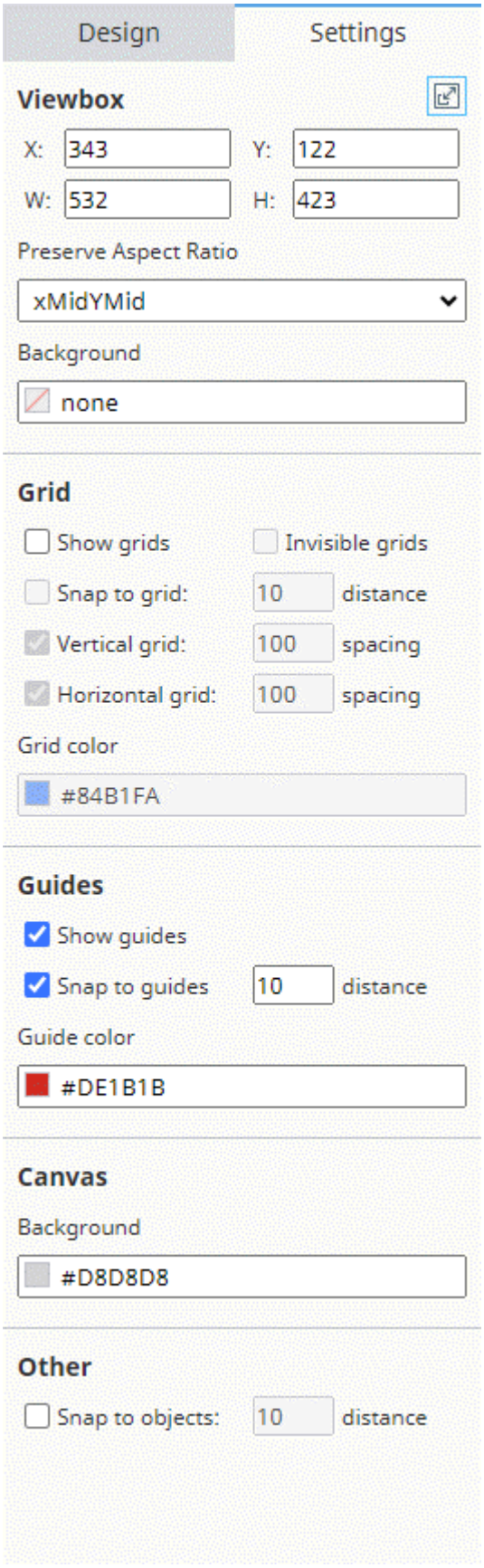
Stroke Configuration Example

The following example displays applied Start Type, Dash Array, and Cap stroke properties for a **perspective-arc**, where the stroke fill is `#000000` and `1px` wide.

Property	Value
Start Type	Square
Start Scale	4
Start Rotate	Auto
Dash Array	1, 4
Cap	Round

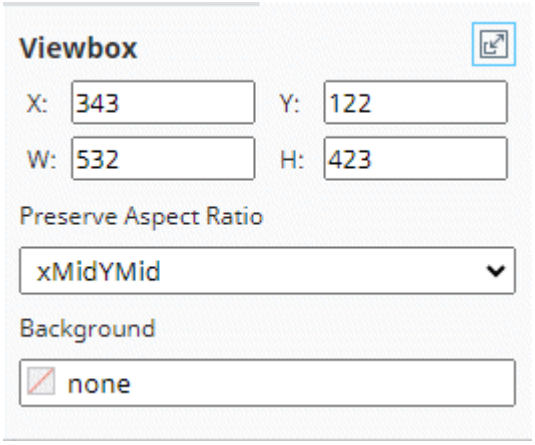


Settings Panel



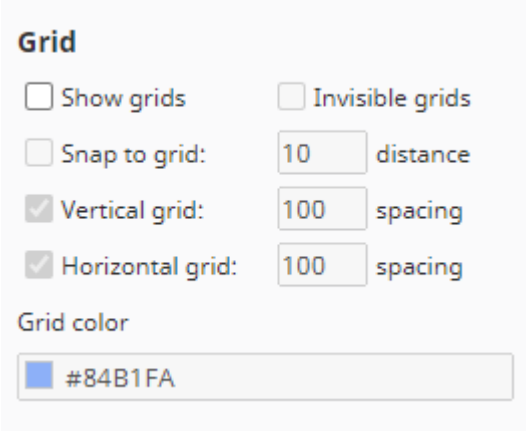
The **Settings Panel** in the Drawing Editor contains configuration options to control the viewBox, grids, guides, and other Drawing component properties. The panel is divided into several subsections:

Viewbox Properties



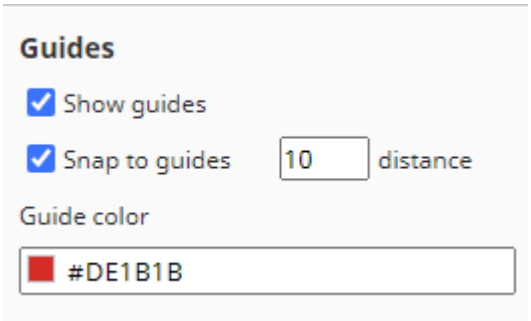
Property	Description	Property Type
Resize Viewbox to Content	Resizes the viewbox to fit the content inside.	value: boolean
X	The x-coordinate of the viewbox's origin.	value: numeric
Y	The y-coordinate of the viewbox's origin.	value: numeric
W	The width of the viewbox.	value: numeric
H	The height of the viewbox.	value: numeric
Preserve Aspect Ratio	Specifies how the viewBox's aspect ratio is maintained.	value: string
Background	The background color of the viewbox.	value: color

Grid Properties



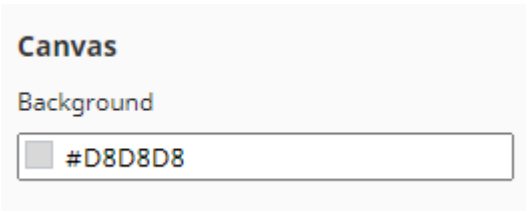
Property	Description	Property Type
Show Grids	Toggles the visibility of grids on the canvas.	value: boolean
Invisible Grids	Enables invisible grids (snap to grids without showing them).	value: boolean
Snap to Grid	Enables snapping objects to grid lines.	value: boolean
Distance	Sets the snapping distance for grids.	value: numeric
Vertical Grid	Enables the vertical grid.	value: boolean
Vertical Grid Spacing	Sets the spacing for vertical grid lines.	value: numeric
Horizontal Grid	Enables the horizontal grid.	value: boolean
Horizontal Grid Spacing	Sets the spacing for horizontal grid lines.	value: numeric
Grid Color	Sets the color of the grid lines.	value: color

Guides Properties



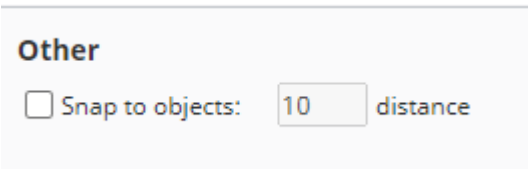
Property	Description	Property Type
Show Guides	Toggles the visibility of guides on the canvas.	value: boolean
Snap to Guides	Enables snapping objects to guide lines.	value: boolean
Distance	Sets the snapping distance for guides.	value: numeric
Guide Color	Sets the color of the guide lines.	value: color

Canvas Properties



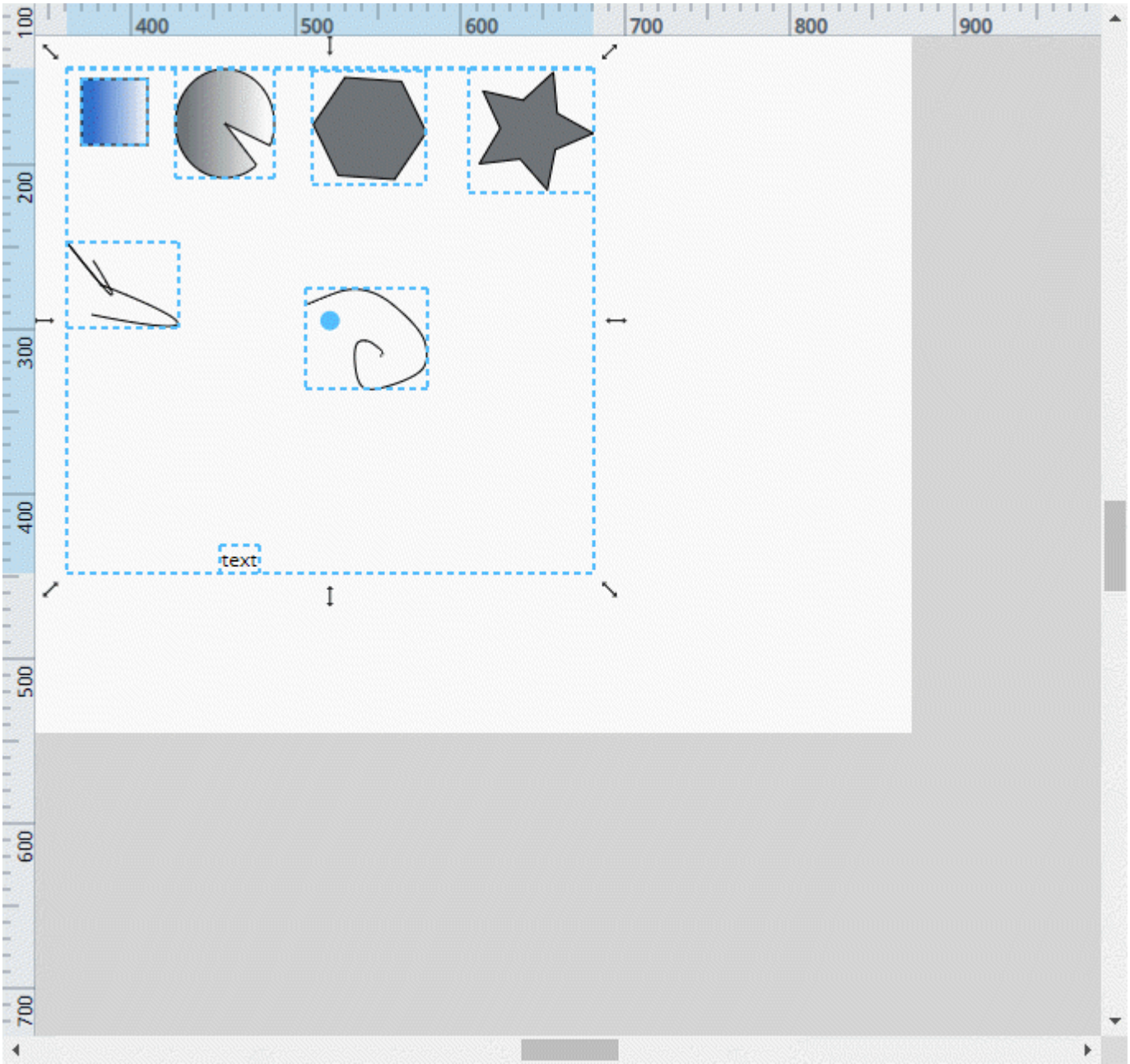
Property	Description	Property Type
Background	Sets the background color of the canvas.	value: color

Other Properties



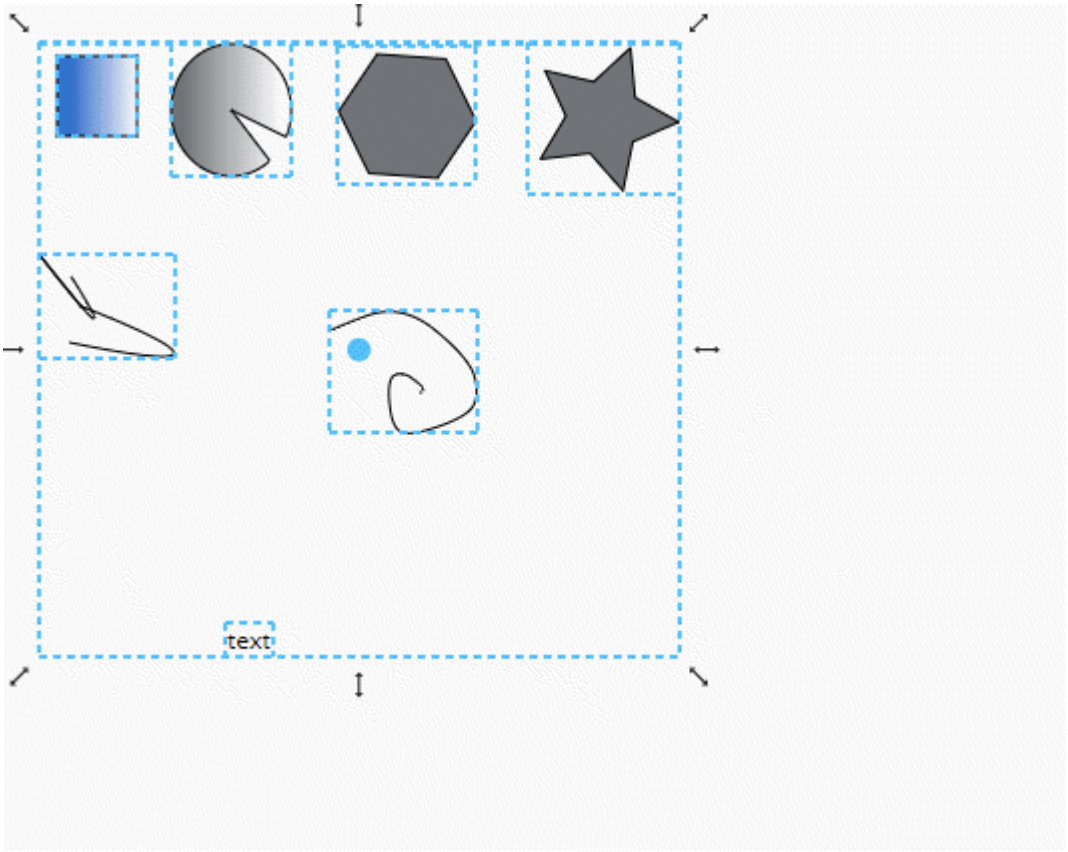
Property	Description	Property Type
Snap to Objects	Enables snapping objects to other objects.	value: boolean
Distance	Sets the snapping distance for object snapping.	value: numeric

Viewbox



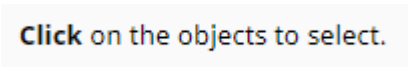
The **Viewbox** defines the visible area of the Canvas, establishing a coordinate system for your drawing. This controls the scaling and positioning of objects. The **Viewbox** can be adjusted to display a specific portion of the Canvas, and it is particularly useful when working with large or detailed drawings. You can adjust the **Viewbox** properties in the Settings Panel to resize, reposition, and preserve the aspect ratio of the drawing content.

Canvas



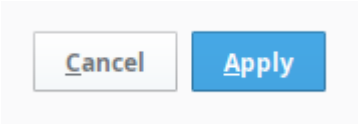
The **Canvas** is the primary workspace where you will create and edit vector drawings. The **Canvas** provides infinite space for designing and arranging elements.

Status Bar



The **Status Bar** displays helpful information such as the current mouse position, zoom level, and contextual tips depending on the active tool and editing state. This information is updated in real-time as you work on your drawing.

Closing the Editor



At the bottom of the Drawing Editor, you'll find the **Apply** and **Cancel** buttons. When you're finished with your edits, click **Apply** to save your changes to the drawing. If you do not want to save your changes, click **Cancel** to discard them.

You can also close the Drawing Editor by clicking the close button in the upper-right corner. If there are unsaved changes, a prompt will appear asking whether to apply the changes or discard them.

Perspective - Drawing Scripting

This page details the various scripting, component, and extension functions available for Perspective's **Drawing** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onElementClicked

New in 8.3.5 This event is fired when an element in the drawing has been clicked.

event.elementAddress

- Object Path
  - event.elementAddress
- Type
  - List
- Description
  - An array of numbers indicating where the clicked element is in the elements tree. For example, [0,2,1] would be the second child of the third child of the first element.

event.elementName

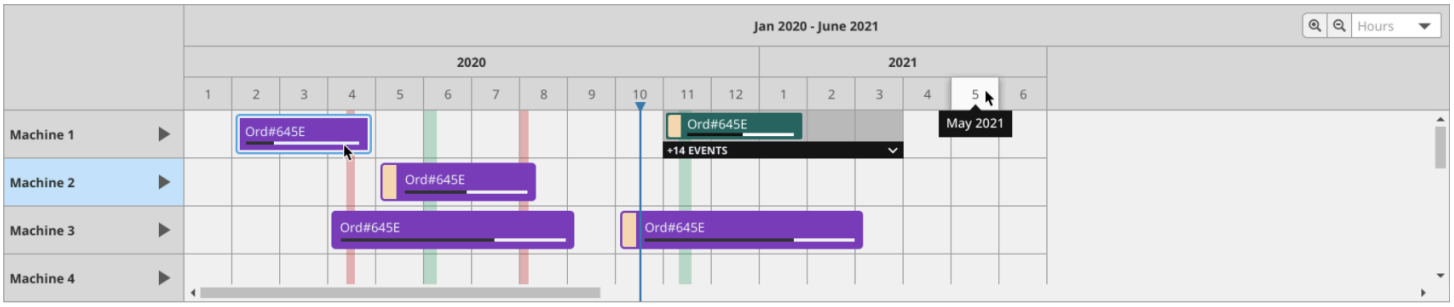
- Object Path
  - event.elementName
- Type
  - String

- Description
    - The name attribute of the element that was clicked.

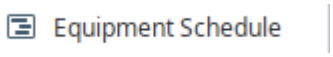
event.event

- Object Path
    - event.event
- Type
    - Dictionary
- Description
    - The event object generated from the mouse click.

Perspective - Equipment Schedule



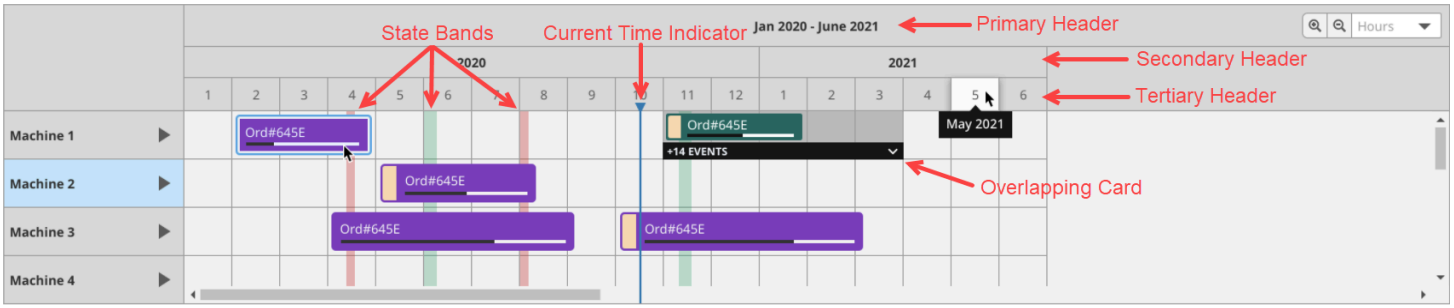
Component Palette Icon:

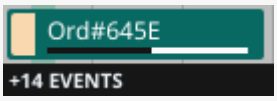
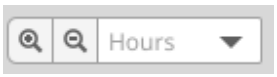


Description

The Equipment Schedule component is a mix between the Status Chart, Gantt Chart, and Calendar components. It conveys equipment scheduling information in a concise and easily digestible format.

Interface Elements



Icon	Element	Description
	Current Time Indicator	Marker that indicates the current time, according to the session's timestamp. Style can be configured via currentTimeIndicator.
	State Bands	Bands indicating downtime and break events. Styles can be configured via downtimeEventStyle and breakEventStyle.
	Standard Card	A card on the equipment schedule corresponds to a scheduled event. During a Perspective Session, cards can be moved and resized by clicking and dragging.
	Overlapping Card	When more than one event is scheduled during the same time period for one item, an overlapping card will represent all events scheduled during that time period. Users can click the black drop down bar to select and edit individual events within the overlapping card.
	Zoom level	Sets the zoom level for the chart. Levels can be selected from the drop down menu or by clicking the magnifying glasses to zoom in or out. Valid values include month, day, 12-hr, 8-hr, 6-hr, 3-hr,15-min, hours, and minutes.
	Add New Event	Clicking within any empty square on the equipment schedule will allow the user to add a new event onto the chart. Note that **Component Events** must be configured and the addEnabled property must be true.
	Primary Header	Header that represents the full range of time displayed on the chart. Styles can be configured via headerStyles.primaryHeaderStyle.
	Secondary Header	Header that represents subsets of time displayed in the Primary Header. Styles can be configured via headerStyles.secondaryHeaderStyle.
	Tertiary Header	Header that represents subsets of time displayed in the Secondary Header. Styles can be configured

Icon	Element	Description
		via headerStyles.tertiaryHeaderStyle.

Properties

Name	Description	Data Type
addEnabled	If enabled, users can add events to an item's schedule. Note that Component Events must be configured to implement the desired functionality when the events are fired.	boolean
resizeEnabled	If enabled, users can resize events. Note that Component Events must be configured to implement the desired functionality when the events are fired.	boolean
moveEnabled	If enabled, users can move events. Note that Component Events must be configured to implement the desired functionality when the events are fired.	boolean
deleteEnabled	If enabled, users can delete events. Note that Component Events must be configured to implement the desired functionality when the events are fired.	boolean
items	The cells, or equipment items, with schedules displayed on the chart. Each row on the equipment schedule corresponds to one item. • **id**: The ID for the item. • **label**: The label for the item displayed on the equipment schedule. Value is a string. • **iconConfig**: ◦ **path**: Image path to the icon. Value is string. ◦ **color**: The color of the icon. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. Value is a string. ◦ **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **headerStyle**: Sets a style for the header. Full menu of **style options** is available for text, background, margin	array

Name	Description	Data Type
	and padding, border, shape and miscellaneous. You can also specify a **style class**. • **rowStyle**: Sets a style for the row. Will override default styles set by the component's rowStyle property. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
dateRange	• **startDate**: The beginning of the time range to display. Value is DateTime. • **endDate**: The end of the time range to display. Value is DateTime.	object
defaultZoom	Default zoom level for the chart. Valid values include month, day, 12-hr, 8-hr, 6-hr, 3-hr, 15-min, hours, and minutes.	value: string
rowHeight	Sets the height for all rows on the equipment schedule.	value: numeric
scheduledEvents	The scheduled events for all configured items. • **itemId**: The item identifier where this event is scheduled. • **eventId**: The event identifier of the scheduled event. The ID must be unique. • **startDate**: The start date of the scheduled event. Value is DateTime. • **endDate**: The end date of the scheduled event. Value is DateTime. • **label**: The label for the event displayed on the equipment schedule. Value is string. • **leadTime**: The amount of lead time to display on the equipment schedule before the scheduled event's startDate. Value is numeric (in seconds). • **leadStyle**: Sets a style for the leadTime. Full menu of **style options** is available for text, background, margin	object

Name	Description	Data Type
	and padding, border, shape and miscellaneous. You can also specify a **style class**. • **percentDone**: If progressBar is enabled, this value will be displayed in the event's progress bar. Value is numeric. • **style**: Sets a style for the event. Will override styles set by scheduledEventStyle. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
downtimeEvents	Downtime events correlated to a specific item.	array
breakEvents	Scheduled breaks, which will appear as downtime for all items.	array
selectedEvent	• **itemId**: The ID of the selected item. • **eventId**: The ID of the selected event.	object
progressBar	• **enabled**: If enabled, scheduled events will display a progress bar. Value is boolean. • **bar**: ◦ **color**: The color of the progress bar. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector.**. Value is string. ◦ **style**: Sets a style for the progress bar. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **track**: ◦ **color**: The color of the progress bar track. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector.**. Value is string.	object

Name	Description	Data Type
	◦ **style**: Sets a style for the progress bar track. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **valueDisplay**: ◦ **enabled**: If enabled, the progress bar will display percentDone as a numeric value. Value is boolean. ◦ **format**: Sets a numeric format for the valueDisplay. Valid values include none, integer, percent, currency, and duration. Value is string. ◦ **justify**: Sets alignment for the valueDisplay on the progress bar. Valid values include left, center, and right. Value is string. ◦ **style**: Sets a style for the value display. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
currentTimeIndicator	• **color**: The color of the current time indicator. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector.**. Value is string. • **opacity**: The opacity level for the current time indicator. Value is numeric. • **width**: The width of the current time indicator. Value is numeric.	object
headerStyles	• **primaryHeaderStyle**: Sets a style for the primary header. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **secondaryHeaderStyle**: Sets a style for the secondary header. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Name	Description	Data Type
	• **tertiaryHeaderStyle**: Sets a style for the tertiary header. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
rowStyle	Sets a style for all rows. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
selectedEventStyle	Sets a style for the selected event. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
scheduledEventStyle	Sets a style for scheduled events. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
downtimeEventStyle	Sets a style for downtime events. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
breakEventStyle	Sets a style for break events. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
bodyStyle	Sets a style for the body. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
style	Sets a style for the component. Full menu of **style options** is available for text, background, margin and padding, border,	object

Name	Description	Data Type
	shape and miscellaneous. You can also specify a **style class**.	

Scripting

See the **Perspective - Equipment Schedule Scripting** page for the full list of scripting functions available for this component.

Perspective - Equipment Schedule Scripting

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onAddEvent

Event is fired after a user adds an event to the schedule.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.end

- Object Path
  - event.end
- Type
  - Integer or float
- Description
  - The end date of the new event.

event.itemId

- Object Path

    - event.itemId
- Type
    - String, integer, or float
- Description
    - The item identifier where this event was created.

event.start

- Object Path
    - event.start
- Type
    - Integer or float
- Description
    - The start date of the new event.

Example

```
from random import random, randint

format = "yyyy-MM-dd HH:mm:ss z"
locale = "en-us"
scheduled = self.props.scheduledEvents
r = random() * 255
g = random() * 255
b = random() * 255
color = "rgb(" + str(r) + "," + str(g) + "," + str(b) + ")"
percentDone = randint(0, 100)
leadTime = randint(300, 6000)

item = {
    "endDate": system.date.parse(event.end, format, locale),
    "itemId": event.itemId,
    "startDate": system.date.parse(event.start, format, locale),
    "eventId": "event_" + str(randint(1000, 10000)),
    "backgroundColor": color,
    "percentDone": percentDone,
    "leadTime": leadTime
}
scheduled.append(item)
```

onMoveEvent

Event is fired after a user moves an event.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.end

- Object Path
  - event.end
- Type
  - Integer or float

- Description
    - The new end date of the moved event.

event.eventId

- Object Path
    - event.eventId
- Type
    - String, integer, or float
- Description
    - The event identifier of the moved event.

event.itemId

- Object Path
    - event.itemId
- Type
    - String, integer, or float
- Description
    - The item identifier where this event was moved.

event.start

- Object Path
    - event.start
- Type
    - Integer or float
- Description
    - The new start date of the moved event.

Example

```
format = "yyyy-MM-dd HH:mm:ss z"
locale = "en-us"
scheduled = self.props.scheduledEvents
for count, toFind in enumerate(scheduled):
    if toFind.eventId == event.eventId and event.itemId == toFind.itemId:
        toFind.startDate = system.date.parse(event.start, format, locale)
        toFind.endDate = system.date.parse(event.end, format, locale)
```

onResizeEvent

Event is fired after a user resizes an event.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.end

- Object Path
  - event.end
- Type
  - Integer or float
- Description
  - The new end date of the resized event.

event.eventId

- Object Path
  - event.eventId
- Type

- String, integer, or float

- Description
  - The event identifier of the resized event.

event.itemId

- Object Path
  - event.itemId
- Type
  - String, integer, or float
- Description
  - The item identifier where this event was resized.

event.start

- Object Path
  - event.start
- Type
  - Integer or float
- Description
  - The new start date of the resized event.

Example

```
format = "yyyy-MM-dd HH:mm:ss z"
locale = "en-us"
scheduled = self.props.scheduledEvents
for count, toFind in enumerate(scheduled):
    if toFind.eventId == event.eventId and event.itemId == toFind.itemId:
        toFind.startDate = system.date.parse(event.start, format, locale)
        toFind.endDate = system.date.parse(event.end, format, locale)
```

onDeleteEvent

Event is fired after a user deletes an event.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.end

- Object Path
  - event.end
- Type
  - Integer or float
- Description
  - The end date of the deleted event.

event.eventId

- Object Path
  - event.eventId
- Type
  - String, integer, or float
- Description
  - The event identifier of the deleted event.

event.itemId

- Object Path
  - event.itemId

- Type
  - String, integer, or float
- Description
  - The item identifier where this event was deleted.

event.start

- Object Path
  - event.start
- Type
  - Integer or float
- Description
  - The start date of the deleted event.

Example

```
scheduled = self.props.scheduledEvents
format = "yyyy-MM-dd HH:mm:ss z"
locale = "en-us"
for count, toFind in enumerate(scheduled):
    if toFind.eventId == event.eventId and toFind.itemId == event.itemId:
        del scheduled[count]
        break
```

onClickEvent

Event is fired after a user clicks on an event.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.end

- Object Path
  - event.end
- Type
  - Integer or float
- Description
  - The end date of the clicked event.

event.eventId

- Object Path
  - event.eventId
- Type
  - String, integer, or float
- Description
  - The event identifier of the clicked event.

event.itemId

- Object Path
  - event.itemId
- Type
  - String, integer, or float
- Description
  - The item identifier where this event was clicked.

event.start

- Object Path

    - event.start
- Type
    - Integer or float
- Description
    - The start date of the clicked event.

Example

```
scheduled = self.props.scheduledEvents
format = "yyyy-MM-dd HH:mm:ss z"
locale = "en-us"
for count, toFind in enumerate(scheduled):
    if toFind.eventId == event.eventId and toFind.itemId == event.itemId:
        print scheduled[count]
        break
```

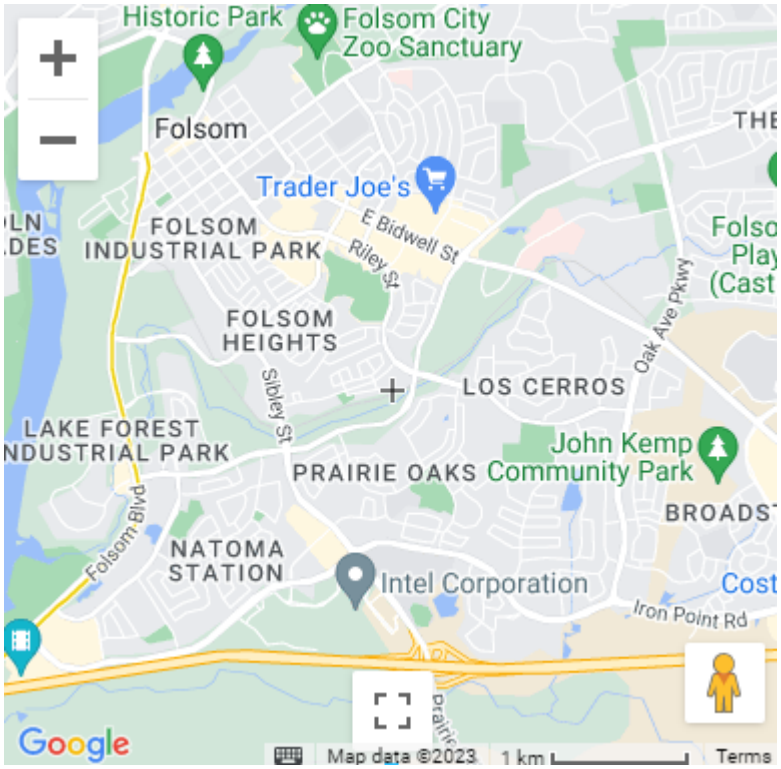
Component Functions

This component does not have component functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Perspective - Google Map



Component Palette Icon:



Description

The Google Map component provides a new interactive map option to the Map component. The Google Map component is based on the LeafletJS plugin to use Google maps basemaps and allows scriptable interactions and various layer configurations including traffic, polygons, markers, and overlays.

> **CAUTION**
>
> A Google Map API Key is required to be entered in the new **googleMapsApiKey** session prop to provide a functional map without a Development Mode watermark. This API key will be exposed on the web application, so it is recommended to add restrictions following Google's **API Security Best Practices**.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
init	The map's initial state on load. • **center**: Initial geographic center of the map. ◦ **lat**: Latitudinal coordinate. Default value is 38.660867. ◦ **lng**: Longitudinal coordinate. Default value is -121.159728. • **zoom**: Initial map zoom level. Default is 13.	object
backgroundColor	Color used for the background of the map when the tiles have not yet loaded as the user pans.	value: string
clickableIcons	Allow map icons to be clickable. These icons are also known as points of interest (POI) and represent areas like parks, schools, and shops . Default is true.	value: boolean
cursor	Cursor related configuration. • **draggable**: The name or URL of the cursor to display when mousing over a draggable map. This property uses the CSS cursor attribute to change the icon. As with the CSS property, you must specify at least one fallback cursor that is not a URL. Value is string. • **dragging**: The name or URL of the cursor to display when the map is being dragged. This property uses the CSS cursor attribute to change the icon. As with the CSS property, you must specify at least one fallback cursor that is not a URL. Value is string.	object
controlSize	Size in pixels of the controls appearing on the map.	value: numeric
disableDefaultUI	Enables/Disables all default UI buttons. May be overridden individually. Does not disable the keyboard controls, which are separately controlled by the **keyboardShortcuts** property. Does not disable	value: boolean

Name	Description	Property Type
	gesture controls, which are separately controlled by the **gestureHandling** property.	
fullscreen	Fullscreen related configuration. • **controls**: Enabled/Disabled state of the Fullscreen control. Default is enabled. Value is boolean. • **controlPosition**: Control display position. Default is top_left. Value is string.	object
gestureHandling	Controls how the map handles gestures. Options include cooperative, greedy, auto, and none. Default is set to auto.	value: string
heading	The heading for aerial imagery in degrees measured clockwise from cardinal direction North.	value: numeric
isFractionalZoomEnabled	Whether the map should allow fractional zoom levels.	value: boolean
keyboardShortcuts	Allows the map to be controlled by the keyboard when set to true.	value: boolean
layers	Map layers configuration. See the **Layers** section below for more detail on the available map layers.	object
mapID	The Map ID of the map.	value: string
mapType	MapType related configuration. • **id**:The initial Map mapTypeId. Value is string. • **controls**: The initial enabled/disabled state of the map type control. Value is boolean. • **controlPosition**: Control display position. Default is top_left. Value is string. • **controlMapTypeIds**: IDs of map types to show in the control.	object

Name	Description	Property Type
	• **controlStyle**: Style of map type control to display. Value is string.	
restriction	Defines a boundary that restricts the area of the map accessible to users. When set, a user can only pan and zoom while the camera view stays inside the limits of the boundary. • **bounds**: When set, a user can only pan and zoom inside the given bounds. Bounds can restrict both longitude and latitude, or can restrict latitude only. For latitude-only bounds, use west and east longitudes of -180 and 180, respectively. For example: north: northLat, south: southLat, west: -180, east: 180. ◦ **north**: Latitude for Northern boundary. Value is string. ◦ **east**: Longitude for the Eastern boundary. Value is string. ◦ **south**: Latitude for the Southern boundary. Value is string. ◦ **west**: Longitude for the Western boundary. Value is string. • **strict**: Bounds can be made more restrictive by setting the strictBounds flag to true. This reduces how far a user can zoom out, ensuring that everything outside of the restricted bounds stays hidden. The default is false, meaning that a user can zoom out until the entire bounded area is in view, possibly including areas outside the bounded area. Value is boolean.	object

Name	Description	Property Type
rotate	Rotate related configuration. • **controls**: Enabled/Disabled state of the Rotate control. Default is enabled. Value is boolean. • **controlPosition**: Control display position. Default is top_left. Value is string.	object
scale	Scale related configuration. • **controls**: Enabled/Disabled state of the Scale control. Default is enabled. Value is boolean.	object
tilt	Controls the automatic switching behavior for the angle of incidence of the map. The only allowed values are 0 and 45.	value: numeric
zoom	Zoom related configuration. • **controls**: Enabled/Disabled state of the Zoom control. Default is enabled. Value is boolean. • **controlPosition**: Control display position. Default is top_left. Value is string. • **max**: The maximum zoom level which will be displayed on the map. Value is numeric. • **min**: The minimum zoom level which will be displayed on the map. Value is numeric.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Layers Properties

overlayView

Renders instances of Perspective views within the active project.

Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	Enable/disable rendering of the OverlayView layer.	value: boolean
mapPaneName	Map pane in which to render the OverlayView. Options include floatPane, mapPane, markerLayer, overlayLayer, overlayMouseTarget.	value: string
bounds	Sets the bounds of the overlay. Either of bounds or position are required. • **corner1**: South-West corner of the overlayView. ◦ **lat**: Latitudinal coordinate. Value is numeric. ◦ **lng**: Longitude coordinate. Value is numeric. • **corner2**: North-East corner of the overlayView. ◦ **lat**: Latitudinal coordinate. Value is numeric. ◦ **lng**: Longitude coordinate. Value is numeric.	object
position	OverlayView position. Either of bounds or positions are required.	object
views	OverlayView instance configuration. • **path**: Path to the view instance. Value is string. • **params**: Params to pass to the view instance. • **properties**: Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers. • **style**: Sets a style for the selected view. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.	object

Name	Description	Property Type
transparentBackground	If enabled, disables the default background color.	value: boolean
shadow	If enabled, apply a box shadow around the view.	value: boolean
zIndex	The z-index of the layer.	value: numeric
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object
style	Sets a style for this overlayView layer. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

groundOverlay

Binds and scales an image to fit projected ground coordinates.

Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	Enable/disable rendering of GroundOverlay layer.	value: boolean
overlays	Overlay related configuration. • **source**: The URL of the image to display. Value is string. • **bounds**: Sets the bounds of the GroundOverlay. ◦ **corner1**: South-West corner of the GroundOverlay, set using `lat` and `lng` properties. ◦ **corner2**: North-East corner of the GroundOverlay, set using `lat` and `lng` properties. • **properties**: Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object
clickable	If true, the ground overlay can receive mouse events.	value: boolean
opacity	The opacity of the overlay, expressed as a number between 0 and 1. Optional.	value: numeric
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object

marker

A marker identifies a location on a map. By default, a marker uses a standard image. Markers can display custom images.

Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	Enable/disable rendering of the marker layer.	value: boolean
visible	If true, the marker is visible.	value: boolean
markers	Marker related configuration. • **position**: Marker position configuration. ◦ **lat**: Latitudinal coordinate. Value is numeric. ◦ **lng**: Longitude coordinate. Value is numeric. • **icon**: Marker icon configuration. ◦ **type**: The icon type used as to represent the marker. Options include default, image, and symbol. Value is string. • **popup**: If enabled, configured popups can be made visible for this marker. Popups do not apply to clusters. • **tooltip**: If enabled, configured tooltips can be made visible for this marker. Tooltips do not apply to clusters. • **properties**: Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object
clusterMarkers	Enables marker clustering.	value: boolean
animation	Starts an animation. Any ongoing animation will be cancelled. Passing in null will cause any animation to stop. Default options include none, bounce, and drop.	value: string
clickable	If true, the marker receives mouse and touch events.	value: boolean

Name	Description	Property Type
crossOnDrag	If false, disables cross that appears beneath the marker when dragging.	value: boolean
cursor	Mouse cursor to show on hover.	value: string
draggable	If true, the marker can be dragged.	value: boolean
icon	Marker icon configuration. • **type**: Selects shape configuration. Options include default, image, and symbol. Value is string.	object
label	Adds a label to the marker.	value: string
opacity	The marker's opacity between 0.0 and 1.0.	value: numeric
optimized	Optimization enhances performance by rendering many markers as a single static element. This is useful in cases where a large number of markers is required.	value: boolean
popup	If enabled, popup configuration for this marker. Popups do not apply to clusters. • **enabled**: Enable marker popup. Value is boolean. • **visible**: If true, the popup is visible. Value is boolean. • **content**: The popup content to display. ◦ **text**: Text to display. Value is numeric. ◦ **view**: A view to display as popup content. If configured, overrides the text property. ▪ **path**: Path of view to display. Value is string. ▪ **params**: Parameters for the view. If passing parameters into the embedded view, the names here must match the parameters on that view. • **autoClose**: Set it to false if you want to override the default behavior of the popup closing when another	object

Name	Description	Property Type
	popup is opened. Value is boolean. • **width**: Width of the popup. Min and Max values allowed. • **disableAutoPan**: Disable panning the map to make the popup fully visible when it opens. Value is boolean. • **pixelOffset**: The offset, in pixels, of the tip of the popup from the x and y points on the map at whose geographical coordinates the popup is anchored. • **arialLabel**: AriaLabel to assign to the popup. Value is string. • **zIndex**: All popups are displayed on the map in order of their z-index, with higher values displaying in front of popups with lower values. By default, popups are displayed according to their latitude, with popups of lower latitudes appearing in front of popups at higher latitudes. Popups are always displayed in front of markers. Value is numeric. • **style**: Sets a style for the popup. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.	
tooltip	If enabled, tooltips can be configured for this marker. Tooltips do not apply to clusters. • **enabled**: Enable marker tooltip. Value is boolean. • **content**: The tooltip content to display. ◦ **text**: Text to display. Value is numeric. ◦ **view**: A view to display as tooltip content. If configured, overrides the text property. ▪ **path**: Path of view to display. Value is string. ▪ **params**: Parameters for the view. If passing parameters into the embedded view, the names here must match the parameters on that view. • **direction**: Direction where to open the tooltip. Possible values are right, left, top, bottom, center, and auto. Auto will dynamically switch between right and left according to the tooltip position on the map. Value is string.	object

Name	Description	Property Type
	• **pixelOffset**: The offset, in pixels, of the tooltip from the marker's position. For tooltips configured with auto direction, positive offsets move the tooltip away from the marker's position. • **permanent**: If true, the tooltip will display permanently, instead of only on mouseover. Value is boolean. • **style**: Sets a style for the tooltip. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class.	
shape	Image map region definition used for drag/click. • **type**: Selects shape configuration. Options include none, circle, poly, rec. Value is string.	object
title	Rollover text.	value: string
zIndex	The z-index of the layer.	value: numeric
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object

circle

An array of circle layers.

Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	Enable/disable rendering of the circle layer.	value: boolean
visible	If true, the circle is visible.	value: boolean
circles	Sets the center of the circle. • **center**: Circle center configuration. ◦ **lat**: Latitudinal coordinate. Value is numeric. ◦ **lng**: Longitude coordinate. Value is numeric. • **fill**: Circle fill configuration. Default properties include enabled, color, and opacity. • **stroke**: Stroke configuration. Default properties include enabled, color, opacity, position, and weight. • **properties**: Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object
radius	Sets the radius of this circle in meters.	value: numeric
clickable	Indicates whether this circle handles mouse events.	value: boolean
draggable	If set to true, the user can drag this circle over the map.	value: boolean
editable	If set to true, the user can edit this circle by dragging the control points shown at the center and around the circumference of the circle.	value: boolean

Name	Description	Property Type
fill	Circle fill configuration. Default properties include enabled, color, and opacity.	object
stroke	Stroke configuration. Default properties include enabled, color, opacity, position, and weight.	object
zIndex	The z-index compared to other polys.	value: string
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object

polygon

Allows the map to draw arbitrary shapes at specified coordinates. An array of polygon layers.

Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	Enable/disable rendering of the polygon layer.	value: boolean
visible	If true, the polygon is visible.	value: boolean
polygons	• **points**: Sets the path for this polygon. The ordered sequence of coordinates that designates a closed loop. Unlike polylines, a polygon may consist of one or more paths. As a result, the paths property may specify one or more arrays of LatLng coordinates. Paths are closed automatically. ◦ **lat**: Latitudinal coordinate. Value is numeric. ◦ **lng**: Longitude coordinate. Value is numeric. • **fill**: Polygon fill configuration. Default properties include enabled, color, and opacity. • **stroke**: Stroke configuration. Default properties include enabled, color, opacity, position, and weight. • **properties**: The z-index compared to other polys. Value is string.	object
clickable	Indicates whether this polygon handles mouse events.	value: boolean
draggable	If set to true, the user can drag this polygon over the map.	value: boolean
editable	If set to true, the user can edit this polygon by dragging the control points shown at the center and around the circumference of the circle.	value: boolean

Name	Description	Property Type
fill	Polygon fill configuration. Default properties include enabled, color, and opacity.	object
geodesic	When true, edges of the polygon are interpreted as geodesic and will follow the curvature of the Earth. When false, edges of the polygon are rendered as straight lines in screen space. Note that the shape of a geodesic polygon may appear to change when dragged, as the dimensions are maintained relative to the surface of the Earth.	value: boolean
stroke	Stroke configuration. Default properties include enabled, color, opacity, position, and weight.	object
zIndex	The z-index compared to other polys.	value: string
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object

polyline

An array of polyline layers.

Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	Enable/disable rendering of the polyline layer.	value: boolean
visible	If true, the polyline is visible.	value: boolean
polylines	• **points**: Sets the path for this polyline. The ordered sequence of coordinates of the Polyline. This path may be specified using either a simple array of LatLngs, or an MVCArray of LatLngs. Note that if you pass a simple array, it will be converted to an MVCArray. Inserting or removing LatLngs in the MVCArray will automatically update the polyline on the map. ◦ **lat**: Latitudinal coordinate. Value is numeric. ◦ **lng**: Longitude coordinate. Value is numeric. • **icons**: The icons to be rendered along the polyline. See the table below for related **child properties**. • **stroke**: Stroke configuration. Default properties include enabled, color, opacity, position, and weight. • **properties**: Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object
clickable	Indicates whether this polyline handles mouse events.	value: boolean
draggable	If set to true, the user can drag this polyline over the map.	value: boolean
editable	If set to true, the user can edit this polyline by dragging the control points shown.	value: boolean

Name	Description	Property Type
geodesic	When true, edges of the polygon are interpreted as geodesic and will follow the curvature of the Earth. When false, edges of the polygone are rendered as straight lines in screen space. Note that the shape of a geodesic polygon may appear to change when dragged, as the dimensions are maintained relative to the surface of the Earth.	value: boolean
stroke	Stroke configuration. Default properties include enabled, color, opacity, position, and weight.	object
zIndex	The zIndex compared to other polys.	value: string
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object

polyline.polylines.icon Properties

Name	Description	Property Type
path	The symbol's path, which is a built-in symbol path, or a custom path expressed using SVG path notation.	value: string
anchor	The position of the symbol relative to the polyline. The coordinates of the symbol's path are translated left and up by the anchor's x and y coordinates respectively.	object
fill	Icon fill configuration. Default properties include enabled, color, and opacity.	object
rotation	The angle by which to rotate the symbol, expressed clockwise in degrees. A symbol in an IconSequence where fixedRotation is false is rotated relative to the angle of the edge on which it lies.	value: numeric
scale	The amount by which the symbol is scaled in size. Defaults to the stroke weight of the polyline. After scaling, the symbol must lie inside a square 22 pixels in size centered at the symbol's anchor.	value: string
stroke	Icon stroke configuration. Default properties include enabled, color, opacity, position, and weight.	object

rectangle

An array of rectangle layers.

Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	Enable/disable rendering of the rectangle layer.	value: boolean
visible	If true, the rectangle is visible.	value: boolean
rectangles	Sets rectangle configuration. • **bounds**: Sets the bounds of the rectangle. ◦ **corner1**: South-West corner of the rectangle, set using `lat` and `lng` properties. ◦ **corner2**: North-East corner of the rectangle, set using `lat` and `lng` properties. • **fill**: Rectangle fill configuration. Default properties include enabled, color, and opacity. • **stroke**: Stroke configuration. Default properties include enabled, color, opacity, position, and weight. • **properties**: Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object
clickable	Indicates whether this rectangle handles mouse events.	value: boolean
draggable	If set to true, the user can drag this rectangle over the map.	value: boolean
editable	If set to true, the user can edit this rectangle by dragging the control points shown.	value: boolean
fill	Rectangle fill configuration. Default properties include enabled, color, and opacity.	object

Name	Description	Property Type
stroke	Stroke configuration. Default properties include enabled, color, opacity, position, and weight.	object
zIndex	The z-index compared to other polys.	value: string
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object

bicycling

Shows bicycle route information.

Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	If true, displays bike lanes and paths and demotes large roads.	value: boolean
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object

traffic

Shows traffic information.

Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	If true, displays current road traffic.	value: boolean
autoRefresh	Whether the traffic layer refreshes with updated information automatically.	value: boolean
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object

transit

Shows transit information.

Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	If true, enables layer that displays transit lines.	value: boolean
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object

kml

Keyhole Markup Language (KML) configuration to specify a set of geographic visualizations.

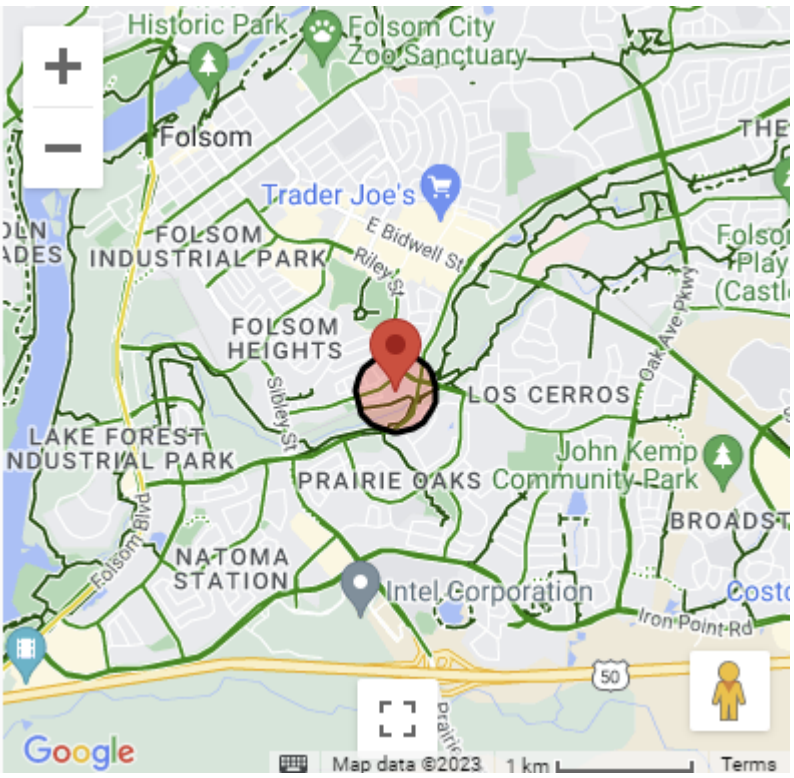
Name	Description	Property Type
name	A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers.	value: string
enabled	Enable/disable rendering of a KML layer.	value: boolean
url	The URL of the KML document to display	object
clickable	If true, the layer receives mouse events	value: boolean
preserveViewport	If this option is set to true or if the map's center and zoom were never set, the input map is centered and zoomed to the bounding box of the contents of the layer.	value: boolean
screenOverlays	Whether to render the screen overlays.	value: boolean
suppressInfoWindows	Suppress the rendering of info windows when layer features are clicked. Default is null.	value: string
zIndex	The z-index of the layer. Default is null.	value: numeric
properties	Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers.	object

Scripting

See the **Perspective - Google Map Scripting** page for the full list of scripting functions available for this component.

Example

The example below incorporates multiple layers to display bike paths and mark a location at the center of an editable circle using layers.



Properties

- layers.marker.0: Default marker icon

Property	Value
markers.0.position.lat	38.660867
markers.0.position.lng	-121.159728
markers.0.icon.type	default

- layers.circle.0: Red circle layer centered at the marker icon location

Property	Value
circles.0.center.lat	38.660867
circles.0.center.lng	-121.159728
circles.0.fill	#FF0000
radius	300

- layers.bicycling: Visible bicycle routes

Property	Value
enabled	true

Perspective - Google Map Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Google Map** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

MapBounds Events

onMapBoundsChanged

Called when the viewport bounds of the map have changed. Returns the north, south, east, and west bound points.

Object Path	Type	Description
event.north	Numeric	The latitude value for the Northern boundary.
event.east	Numeric	The longitude value for the Eastern boundary.
event.south	Numeric	The latitude value for the Southern boundary.
event.west	Numeric	The longitude value for the Western boundary.

onMapCenterChanged

Called when the center of the map has changed. Returns the new map center as lat and lng.

Object Path	Type	Description
event.lat	Numeric	The latitude value for the center of the map after being changed.
event.lng	Numeric	The longitude value for the center of the map after being changed.

Map Events

onMapClick

Interaction event fired when the map is clicked. Returns the lat and lng of the mouse click as it translates on the map.

Object Path	Type	Description
event.lat	Numeric	The latitude of where on the map the user clicked.
event.lng	Numeric	The longitude of where on the map the user clicked.

onMapDoubleClick

Interaction event fired when the map is double-clicked. Returns the lat and lng of the mouse double-click as it translates on the map.

Object Path	Type	Description
event.lat	Numeric	The latitude of where on the map the user double-clicked.
event.lng	Numeric	The longitude of where on the map the user double-clicked.

onMapDrag

Interaction event fired while the user drags the map. Returns the lat and lng of the map center while moving.

Object Path	Type	Description
event.lat	Numeric	The latitude of the map center while the user moves the map.
event.lng	Numeric	The longitude of the map center while the user moves the map.

onMapDragEnd

Interaction event fired when the user stops dragging the map. Returns the lat and lng of the new map center.

Object Path	Type	Description
event.lat	Numeric	The latitude of the map center after the map movement is over.
event.lng	Numeric	The longitude of the map center after the map movement is over.

onMapDragStart

Interaction event fired when the user starts dragging the map. Returns the lat and lng of the starting map center.

Object Path	Type	Description
event.lat	Numeric	The latitude of the map center when map dragging begins.
event.lng	Numeric	The longitude of the map center when map dragging begins.

onMapHeadingChanged

Interaction event fired when the map heading property changes. Returns the new map heading value.

Object Path	Type	Description
event.heading	Numeric	The heading value of the map.

onMapMouseMove

Interaction event fired when the user's mouse moves over the map container. Returns the lat and lng of the mouse location on the map.

Object Path	Type	Description
event.lat	Numeric	The latitude of the mouse location on the map.
event.lng	Numeric	The longitude of the mouse location on the map.

onMapMouseOut

Interaction event fired when the user's mouse exits the map container.

Object Path	Type	Description
event.lat	Numeric	The latitude of the mouse location when exiting the map.
event.lng	Numeric	The longitude of the mouse location exiting the map.

onMapMouseOver

Interaction event fired when the user's mouse enters the map container.

Object Path	Type	Description
event.lat	Numeric	The latitude of the mouse location when entering the map.
event.lng	Numeric	The longitude of the mouse location entering the map.

onMapResize

Interaction event fired when the map size has changed. Returns the width and height of the new map size.

Object Path	Type	Description
event.width	Numeric	The width value of the map.
event.height	Numeric	The height value of the map.

onMapRightClick

Interaction event fired when the DOM contextmenu event is fired on the map container.

Object Path	Type	Description
event.lat	Numeric	The latitude of the mouse location.
event.lng	Numeric	The longitude of the mouse location.

onMapTiltChanged

Interaction event fired when the map tilt property changes. Returns the new tilt value.

Object Path	Type	Description
event.tilt	Numeric	The tilt value of the map. Value will be null, 0, or 45.

onMapTypeIdChanged

Interaction event fired when the map type ID property changes. Returns the new map type ID value.

Object Path	Type	Description
event.mapTypeId	String	The mapTypeId value of the map.

onMapZoomChanged

Interaction event fired when the map zoom property changes. Returns the new zoom level.

Object Path	Type	Description
event.zoom	Numeric	The zoom level of the map view.

BicylingLayer Events

onBicyclingLayerLoad

Interaction event fired when the bicycling layer loads.

Object Path	Type	Description
event.name	String	The name of the bicycling layer.

onBicyclingLayerUnmount

Interaction event fired when the bicycling layer unloads.

Object Path	Type	Description
event.name	String	The name of the bicycling layer.

Circle Events

onCircleCenterChanged

Interaction event fired when the center of the circle changes.

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude value of the center of the circle.
event.lng	Numeric	The longitude value of the center of the circle.

onCircleClick

Interaction event fired when the circle is clicked.

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of where on the circle the user clicked.
event.lng	Numeric	The longitude of where on the circle the user clicked.

onCircleDoubleClick

Interaction event fired when the circle is double-clicked.

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of where on the circle the user double-clicked.
event.lng	Numeric	The longitude of where on the circle the user double-clicked.

onCircleDrag

Interaction event repeatedly fired while the user drags the circle.

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of the map center while the user moves the map.
event.lng	Numeric	The longitude of the map center while the user moves the map.

onCircleDragEnd

Interaction event fired when the user stops dragging the circle.

Object Path	Type	Description
event.lat	Numeric	The latitude of the map center while the user moves the map.
event.lng	Numeric	The longitude of the map center while the user moves the map.

onCircleDragStart

Interaction event fired when the user starts dragging the circle.

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of the map center when map dragging begins.
event.lng	Numeric	The longitude of the map center when map dragging begins.

onCircleLoad

Interaction event fired when the circle layer loads.

Object Path	Type	Description
event.name	String	The name of the circle layer.

onCircleMouseDown

Interaction event fired on the mousedown of the circle

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of the mouse location on the circle layer.
event.lng	Numeric	The longitude of the mouse location on the circle layer.

onCircleMouseMove

Interaction event fired when the mousemove of the circle

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of the mouse location on the circle layer.
event.lng	Numeric	The longitude of the mouse location on the circle layer.

onCircleMouseOut

Interaction event fired when the mouseout of the circle

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of the mouse location on the circle layer.
event.lng	Numeric	The longitude of the mouse location on the circle layer.

onCircleMouseOver

Interaction event fired when the mouseover of the circle

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of the mouse location on the circle layer.
event.lng	Numeric	The longitude of the mouse location on the circle layer.

onCircleMouseUp

Interaction event fired when the mouseup of the circle

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of the mouse location on the circle layer.
event.lng	Numeric	The longitude of the mouse location on the circle layer.

onCircleRadiusChanged

Interaction event fired when the radius of the circle is changed.

Object Path	Type	Description
event.name	String	The name of the circle.
event.radius	Numeric	The radius of the circle.

onCircleRightClick

Interaction event fired on right-click of the circle layer.

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of the mouse location on the circle layer.
event.lng	Numeric	The longitude of the mouse location on the circle layer.

onCircleUnmount

Interaction event fired when the circle unloads.

Object Path	Type	Description
event.name	String	The name of the circle.

GroundOverlay Events

onGroundOverlayClick

Interaction event fired when the ground overlay is clicked.

Object Path	Type	Description
event.name	String	The name of the ground overlay layer.
event.lat	Numeric	The latitude of where on the ground overlay the user clicked.
event.lng	Numeric	The longitude of where on the ground overlay the user clicked.

onGroundOverlayDoubleClick

Interaction event fired when the ground overlay is double-clicked.

Object Path	Type	Description
event.name	String	The name of the ground overlay layer.
event.lat	Numeric	The latitude of where on the ground overlay the user double-clicked.
event.lng	Numeric	The longitude of where on the ground overlay the user double-clicked.

onGroundOverlayLoad

Interaction event fired when the ground overlay layer is loaded.

Object Path	Type	Description
event.name	String	The name of the ground overlay layer.

onGroundOverlayUnmount

Interaction event fired when the ground overlay layer unloads.

Object Path	Type	Description
event.name	String	The name of the ground overlay layer.

KmlLayer Events

OnKmlLayerClick

Interaction event fired when a feature in the KML layer is clicked.

Object Path	Type	Description
event.name	String	The name of the KML layer.
event.featuredData	Object	Contains information about the clicked feature.
event.lat	Numeric	The latitude of where the information pop-up window is anchored on the KML layer.
event.lng	Numeric	The longitude of where the information pop-up window is anchored on the KML layer.
event.pixelOffset	Object	The offset to apply to the anchored information pop-up window on the clicked feature.

OnKmlLayerDefaultViewportChanged

Interaction event fired when the KML layer default viewport has changed.

Object Path	Type	Description
event.name	String	The name of the KML layer.

OnKmlLayerLoad

Interaction event fired when the KML layer loads.

Object Path	Type	Description
event.name	String	The name of the KML layer.

OnKmlLayerStatusChanged

Interaction event fired when the KML layer has finished loading.

Object Path	Type	Description
event.name	String	The name of the KML layer.

OnKmlLayerUnmount

Interaction event fired when the KML layer unloads.

Object Path	Type	Description
event.name	String	The name of the KML layer.

Marker Events

onMarkerClick

Interaction event fired when the marker is clicked. Returns the lat and lng of the marker click.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of the marker click.
event.lng	Numeric	The longitude of the marker click.

onMarkerDoubleClick

Interaction event fired when the marker is double-clicked. Returns the lat and lng of the marker double- click.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of where the marker was double-clicked.
event.lng	Numeric	The longitude of where the marker was double-clicked.

onMarkerDrag

Interaction event repeatedly fired while the user drags the marker. Returns the lat and lng of the marker while moving.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of the marker while being dragged.
event.lng	Numeric	The longitude of the marker while being dragged.

onMarkerDragEnd

Interaction event fired when the user stops dragging the marker. Returns the lat and lng of the new marker position.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of the marker after the marker movement is over.
event.lng	Numeric	The longitude of the marker after the marker movement is over.

onMarkerDragStart

Interaction event fired when the user starts dragging the marker. Returns the lat and lng of the starting marker position.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of the marker when the marker dragging begins.
event.lng	Numeric	The longitude of the marker when the marker dragging begins.

onMarkerLoad

Interaction event fired when the marker layer loads.

Object Path	Type	Description
event.name	String	The name of the marker layer.

onMarkerMouseDown

Interaction event fired on the mousedown of the marker.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of the marker location.
event.lng	Numeric	The longitude of the marker location.

onMarkerMouseOut

Interaction event fired on mouseout of the marker.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of the event.
event.lng	Numeric	The longitude of the event.

onMarkerMouseOver

Interaction event fired on mouseover of the marker.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of the event.
event.lng	Numeric	The longitude of the event.

onMarkerMouseUp

Interaction event fired when the mouseup of the marker.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of the event.
event.lng	Numeric	The longitude of the event.

onMarkerPositionChanged

Interaction event fired when the position of the marker changes.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of the new marker position.
event.lng	Numeric	The longitude of the new marker position.

onMarkerRightClick

Interaction event fired on the right-click of the marker.

Object Path	Type	Description
event.name	String	The name of the marker.
event.lat	Numeric	The latitude of the event location.
event.lng	Numeric	The longitude of the event location.

onMarkerUnmount

Interaction event fired when the marker layer unloads.

Object Path	Type	Description
event.name	String	The name of the marker layer.

onMarkerPopupOpened

Interaction event fired when a marker popup opens.

Object Path	Type	Description
event.name	String	The name of the marker layer.

onMarkerPopupClosed

Interaction event fired when a marker popup closes.

Object Path	Type	Description
event.name	String	The name of the marker layer.

OverlayView Events

onOverlayViewLoad

Interaction event fired when the OverlayView layer loads.

Object Path	Type	Description
event.name	String	The name of the OverView layer.

onOverlayViewUnmount

Interaction event fired when the OverlayView layer unloads.

Object Path	Type	Description
event.name	String	The name of the OverView layer.

Polygon Events

onPolygonClick

Interaction event fired when the polygon is clicked. Returns the lat and lng of the mouse click as it translates on the polygon later

Object Path	Type	Description
event.name	String	The name of the polygon layer.
event.lat	Numeric	The latitude of where on the map the user clicked.
event.lng	Numeric	The longitude of where on the map the user clicked.

onPolygonDoubleClick

Interaction event fired when the polygon is double-clicked. Returns the lat and lng of the mouse double-click as it translates on the polygon layer.

Object Path	Type	Description
event.name	String	The name of the polygon layer.
event.lat	Numeric	The latitude of where on the polygon the user double-clicked.
event.lng	Numeric	The longitude of where on the polygon the user double-clicked.

onPolygonDrag

Interaction event repeatedly fired while the user drags the polygon.

Object Path	Type	Description
event.name	String	The name of the polygon layer.
event.lat	Numeric	The latitude of the polygon while the user moves it.
event.lng	Numeric	The longitude of the polygon while the user moves it.

onPolygonDragEnd

Interaction event fired when the user stops dragging the polygon. Returns the lat and lng of the new polygon location.

Object Path	Type	Description
event.name	String	The name of the polygon layer.
event.lat	Numeric	The latitude of the polygon after the polygon movement is over.
event.lng	Numeric	The longitude of the polygon after the polygon movement is over.

onPolygonDragStart

Interaction event fired when the user starts dragging the polygon. Returns the lat and lng of the starting polygon location.

Object Path	Type	Description
event.name	String	The name of the polygon layer.
event.lat	Numeric	The latitude of the polygon when the users starts dragging the polygon.
event.lng	Numeric	The longitude of the polygon when the users starts dragging the polygon.

onPolygonLoad

Interaction event fired when the polygon layer loads.

Object Path	Type	Description
event.name	String	The name of the polygon layer.

onPolygonMouseDown

Interaction event fired on the mousedown of the polygon.

Object Path	Type	Description
event.name	String	The name of the polygon.
event.lat	Numeric	The latitude of the mouse location on the polygon layer.
event.lng	Numeric	The longitude of the mouse location on the polygon layer.

onPolygonMouseMove

Interaction event fired when the user's mouse moves over the polygon layer. Returns the lat and lng of the mouse location on the map.

Object Path	Type	Description
event.name	String	The name of the polygon layer.
event.lat	Numeric	The latitude of the mouse location on the map.
event.lng	Numeric	The longitude of the mouse location on the map.

onPolygonMouseOut

Interaction event fired when the user's mouse exits the polygon layer.

Object Path	Type	Description
event.name	String	The name of the polygon layer.
event.lat	Numeric	The latitude of the mouse location when exiting the map.
event.lng	Numeric	The longitude of the mouse location exiting the map.

onPolygonMouseOver

Interaction event fired when the user's mouse enters the polygon layer.

Object Path	Type	Description
event.name	String	The name of the polygon layer.
event.lat	Numeric	The latitude of the mouse location when entering the polygon.
event.lng	Numeric	The longitude of the mouse location entering the polygon.

onPolygonMouseUp

Interaction event fired when the mouseup of the polygon layer.

Object Path	Type	Description
event.name	String	The name of the polygon layer.
event.lat	Numeric	The latitude of the mouse location on the circle layer.
event.lng	Numeric	The longitude of the mouse location on the circle layer.

onPolygonRightClick

Interaction event fired when the DOM contextmenu event is fired on the polygon layer.

Object Path	Type	Description
event.lat	Numeric	The latitude of the mouse location.
event.lng	Numeric	The longitude of the mouse location.

onPolygonUnmount

Interaction event fired when the polygon layer unloads.

Object Path	Type	Description
event.name	String	The name of the polygon layer.

Polyline Events

onPolylineClick

Interaction event fired when the map is clicked. Returns the lat and lng of the mouse click as it translates on the map.

Object Path	Type	Description
event.lat	Numeric	The latitude of where on the polyline the user clicked.
event.lng	Numeric	The longitude of where on the polyline the user clicked.

onPolylineDoubleClick

Interaction event fired when the map is double-clicked. Returns the lat and lng of the mouse double-click as it translates on the polyline.

Object Path	Type	Description
event.lat	Numeric	The latitude of where on the polyline the user double-clicked.
event.lng	Numeric	The longitude of where on the polyline the user double-clicked.

onPolylineDrag

Interaction event repeatedly fired while the user drags the polyline.

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of the polyline while the user moves the map.
event.lng	Numeric	The longitude of the polyline while the user moves the map.

onPolylineDragEnd

Interaction event fired when the user stops dragging thepolyline. Returns the lat and lng of the new polyline.

Object Path	Type	Description
event.lat	Numeric	The latitude of the polyline center after the map movement is over.
event.lng	Numeric	The longitude of the polyline center after the map movement is over.

onPolylineDragStart

Interaction event fired when the user starts dragging the polyline. Returns the lat and lng of the starting polyline center.

Object Path	Type	Description
event.lat	Numeric	The latitude of the polyline center when map dragging begins.
event.lng	Numeric	The longitude of the polyline center when map dragging begins.

onPolylineLoad

Interaction event fired when the polyline layer loads.

Object Path	Type	Description
event.name	String	The name of the polyline layer.

onPolylineMouseDown

Interaction event fired on the mousedown of the polyline.

Object Path	Type	Description
event.name	String	The name of the polyline.
event.lat	Numeric	The latitude of the mouse location on the polyline layer.
event.lng	Numeric	The longitude of the mouse location on the polyline layer.

onPolylineMouseMove

Interaction event fired when the user's mouse moves over the polyline layer. Returns the lat and lng of the mouse location on the polyline.

Object Path	Type	Description
event.lat	Numeric	The latitude of the mouse location on the polyline.
event.lng	Numeric	The longitude of the mouse location on the polyline.

onPolylineMouseOut

Interaction event fired when the user's mouse exits the polyline.

Object Path	Type	Description
event.lat	Numeric	The latitude of the mouse location when exiting the polyline.
event.lng	Numeric	The longitude of the mouse location exiting the polyline.

onPolylineMouseOver

Interaction event fired when the user's mouse enters the polyline.

Object Path	Type	Description
event.lat	Numeric	The latitude of the mouse location when entering the polyline.
event.lng	Numeric	The longitude of the mouse location entering the polyline.

onPolylineMouseUp

Interaction event fired when the mouseup of the polyline.

Object Path	Type	Description
event.name	String	The name of the circle.
event.lat	Numeric	The latitude of the mouse location on the polyline layer.
event.lng	Numeric	The longitude of the mouse location on the polyline layer.

onPolylineRightClick

Interaction event fired when the DOM contextmenu event is fired on the polyline.

Object Path	Type	Description
event.lat	Numeric	The latitude of the mouse location.
event.lng	Numeric	The longitude of the mouse location.

onPolylineUnmount

Interaction event fired when the polyline layer unloads.

Object Path	Type	Description
event.name	String	The name of the polyline.

Rectangle Events

onRectangleBoundsChanged

Called when the viewport bounds of the rectangle have changed. Returns the north, south, east, and west bound points.

Object Path	Type	Description
event.north	Numeric	The latitude value for the Northern boundary.
event.east	Numeric	The longitude value for the Eastern boundary.
event.south	Numeric	The latitude value for the Southern boundary.
event.west	Numeric	The longitude value for the Western boundary.

onRectangleClick

Interaction event fired when the rectangle is clicked. Returns the lat and lng of the mouse click.

Object Path	Type	Description
event.lat	Numeric	The latitude of where the user clicked.
event.lng	Numeric	The longitude of where the user clicked.

onRectangleDoubleClick

Interaction event fired when the rectangle is double-clicked. Returns the lat and lng of the mouse double-click as it translates on the map.

Object Path	Type	Description
event.lat	Numeric	The latitude of where the user double-clicked.
event.lng	Numeric	The longitude of where the user double-clicked.

onRectangleDrag

Interaction event repeatedly fired while the user drags the rectangle.

Object Path	Type	Description
event.name	String	The name of the rectangle.
event.lat	Numeric	The latitude of the rectangle center while the user moves the rectangle.
event.lng	Numeric	The longitude of the rectangle center while the user moves the rectangle.

onRectangleDragEnd

Interaction event fired when the user stops dragging the rectangle. Returns the lat and lng of the new rectangle center.

Object Path	Type	Description
event.name	String	The name of the rectangle.
event.lat	Numeric	The latitude of the rectangle center while the user moves the rectangle.
event.lng	Numeric	The longitude of the rectangle center while the user moves the rectangle.

onRectangleDragStart

Interaction event fired when the user starts dragging the rectangle. Returns the lat and lng of the starting rectangle center.

Object Path	Type	Description
event.name	String	The name of the rectangle.
event.lat	Numeric	The latitude of the rectangle center while the user moves the rectangle.
event.lng	Numeric	The longitude of the rectangle center while the user moves the rectangle.

onRectangleLoad

Interaction event fired when the rectangle layer loads.

Object Path	Type	Description
event.name	String	The name of the rectangle layer.

onRectangleMouseDown

Interaction event fired on the mousedown of the rectangle.

Object Path	Type	Description
event.name	String	The name of the rectangle.
event.lat	Numeric	The latitude of the mouse location on the rectangle layer.
event.lng	Numeric	The longitude of the mouse location on the rectangle layer.

onRectangleMouseMove

Interaction event fired when the user's mouse moves over the rectangle. Returns the lat and lng of the mouse location.

Object Path	Type	Description
event.name	String	The name of the rectangle.
event.lat	Numeric	The latitude of the mouse location on the rectangle layer.
event.lng	Numeric	The longitude of the mouse location on the rectangle layer.

onRectangleMouseOut

Interaction event fired when the user's mouse exits the rectangle.

Object Path	Type	Description
event.name	String	The name of the rectangle.
event.lat	Numeric	The latitude of the mouse location on the rectangle layer.
event.lng	Numeric	The longitude of the mouse location on the rectangle layer.

onRectangleMouseOver

Interaction event fired when the user's mouse enters the rectangle.

Object Path	Type	Description
event.name	String	The name of the rectangle.
event.lat	Numeric	The latitude of the mouse location on the rectangle layer.
event.lng	Numeric	The longitude of the mouse location on the rectangle layer.

onRectangleMouseUp

Interaction event fired when the mouseup of the rectangle.

Object Path	Type	Description
event.name	String	The name of the rectangle.
event.lat	Numeric	The latitude of the mouse location on the rectangle layer.
event.lng	Numeric	The longitude of the mouse location on the rectangle layer.

onRectangleRightClick

Interaction event fired when the DOM contextmenu event is fired on the rectangle.

Object Path	Type	Description
event.name	String	The name of the rectangle.
event.lat	Numeric	The latitude of the mouse location.
event.lng	Numeric	The longitude of the mouse location.

onRectangleUnmount

Interaction event fired when the KML layer unloads.

Object Path	Type	Description
event.name	String	The name of the rectangle.

TrafficLayer Events

OnTrafficLayerLoad

Interaction event fired when the traffic layer loads.

Object Path	Type	Description
event.name	String	The name of the traffic layer.

OnTrafficLayerUnmount

Interaction event fired when the traffic layer unloads.

Object Path	Type	Description
event.name	String	The name of the traffic layer.

TransitLayer Events

OnTransitLayerLoad

Interaction event fired when the transit layer loads.

Object Path	Type	Description
event.name	String	The name of the transit layer.

OnTransitLayerUnmount

Interaction event fired when the transit layer unloads.

Object Path	Type	Description
event.name	String	The name of the transit layer.

Component Functions

fitBounds

- Description
  - Sets the viewport to contain the given bounds. When the map is set to display none, the fitBounds function reads the map's size as 0x0 and does not do anything.
- Parameters
  - Dictionary **latLngBounds** - A dictionary consisting of two LatLng objects. The LatLng objects combined represent the geographical bounds the map view should be set to.
  - Numeric **padding** - Padding in pixels. The value represents the same padding for all four sides of the map. The bounds will be fit in the part of the map that remains after padding is removed. [optional]
- Return None

panBy

- Description
  - Changes the center of the map by the given distance in pixels. If the distance is less than both the width and height of the map, the transition will be smoothly animated. Note that the map coordinate system increases from west to east (for x values) and north to south (for y values).
- Parameters
  - Numeric **x** - Number of pixels to move the map in the x direction.
  - Numeric **y** - Number of pixels to move the map in the y direction.
- Return
  - None

panTo

- Description
  - Pans the map to a given center. If the change is less than both the width and height of the map, the transition will be smoothly animated.
- Parameters
  - Dictionary **latLng** - The geographic point to pan to.
- Return
  - None

panToBounds

- Description
  - Pans the map by the minimum amount necessary to contain the given LatLngBounds so that the map will be panned to show as much of the bounds as possible inside `{currentMapSizeInPx}` - `{padding}`. The map's zoom, tilt, and heading will not be changed.
- Parameters
  - Dictionary **latLngBounds** - A dictionary consisting of two LatLng objects. The LatLng objects combined represent the geographical bounds the map view will be set to.
  - Numeric **padding** - Padding in pixels. The value represents the same padding for all four sides of the map. [optional]
- Return
  - None

setCenter

- Description
  - Sets the geographical center of the map in latitude and longitude.
- Parameters
  - Dictionary **latLngBounds** - A dictionary consisting of two LatLng objects as `{ lat: number, lng: number }`.
- Return
  - None

setClickableIcons

- Description
  - Controls whether the map icons are clickable or not. A map icon represents a point of interest (POI).
- Parameters
  - Boolean **value** - True to enable clickable map icons, false to disable the clickability of map icons.
- Return
  - None

setHeading

- Description

    - Sets the compass heading for map measured in degrees from cardinal direction North. This method only applies to aerial imagery.
- Parameters
    - Numeric **heading** - The numerical value in degrees to set the compass heading for the map.
- Return
    - None

setMapTypeId

- Description
- Parameters
    - String **mapTypeId** - A string identifier that is used to associate a MapType with a unique value.
- Return
    - None

setTilt

- Description
    - Controls the automatic switching behavior for the angle of incidence of the map. The only allowed values are 0 (default overhead view) and 45. A 45 degree tilt angle will automatically switch to 45 whenever 45° imagery is available for the current zoom level and viewport and switch back when not available.
- Parameters
    - Numeric **tilt** - The numerical value of the tilt angle.
- Return
    - None

setZoom

- Description
    - Sets the zoom of the map.
- Parameters
    - Numeric **zoom** - The numerical value to increase the zoom by. Larger zoom values correspond to a higher resolution.
- Return
    - None

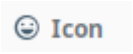
Extension Functions

This component does not have extension functions associated with it.

Perspective - Icon



Component Palette Icon:



Description

The Icon component provides access to a collection of Scalable Vector Graphic (SVG) images, chosen to be useful as icons in a Perspective session.

The materials icon library is a primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. You can also add your own custom repository of icons. For more information on icons, see **Images and Icons in Perspective**.

> **NOTE**
>
> When attempting to change the color of an icon via style class, set the desired color using the Text category's Color property.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
path	Shorthand path to icon source, in format: library/IconName (e.g., material/3d_rotation). See Icons for more information about available icons.	value: string
color	Color of the icon. See **Color Selector**.	color
style	Use Style to customize the visual style of the component. The **Style menu** contains all the tools for modifying text, background, margins, and borders. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1



Property	Value
props.color	#00AC00

Example 2



Property	Value
props.path	material/record_voice_over
props.color	#000088
props.style.borderstyle	ridge
props.style.borderColor	#FFAC47
props.style.borderWidth	6px

Perspective - Image



Component Palette Icon:

Image

Description

The Image component displays either vector or raster format images, such as a jpeg, gif, png, or svg. For examples see **Images and Icons in Perspective**.

When attempting to show images from the **Image Management Tool** on this component, you'll need to prefix /system/images/ to the path. For example:

```
/system/images/Builtin/icons/48/about.png
```

The Image component can also be used to show external images stored relative to the local file system on the client. The file path is similar to having your browser view a local document.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
source	The image source URL. It can be a URL to an image on the internet or Gateway, or even an embedded image. If you use images in the Image Management tool, simply copy their path for this source property, using the format `/system/images/<imagepath>`. For example, the Builtin/icons/16/about.png image path would be **/system/images/Builtin/icons/16/about.png** Additionally, the source can be set to a Base64 encoded image.	value: string
altText	An alternate text for the image if the image cannot be displayed because of a slow connection, an error in the source attribute, if the user uses a screen reader, or some other reason.	value: string
fit	Whether or not the image will size to fit. When in percent mode, the parameters are used to fit based on the percentage of the width and height. When in absolute mode, the image will fit the width and height sizes in pixels. • **mode**: Can be one of the following modes: none, fill, contain, cover, percent, or absolute. Value is string. • **width**: Width of the image in pixels. Value is numeric. • **height**: Height of the image in pixels. Value is numeric. • **scroll**: If false, scrolling is not enabled. Value is boolean.	object
tint	Enables you to tint the entire image a color. • **enabled**: Turn tint on (true) and off (false). Value is boolean. • **color**: If the tint filter is on, this is the color of the tint. See **Color Selector**.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border,	object

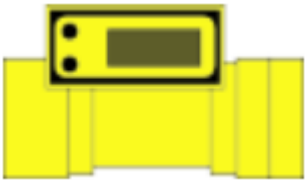
Name	Description	Property Type
	shape and miscellaneous. Classes are predefined styles in a project.	

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1



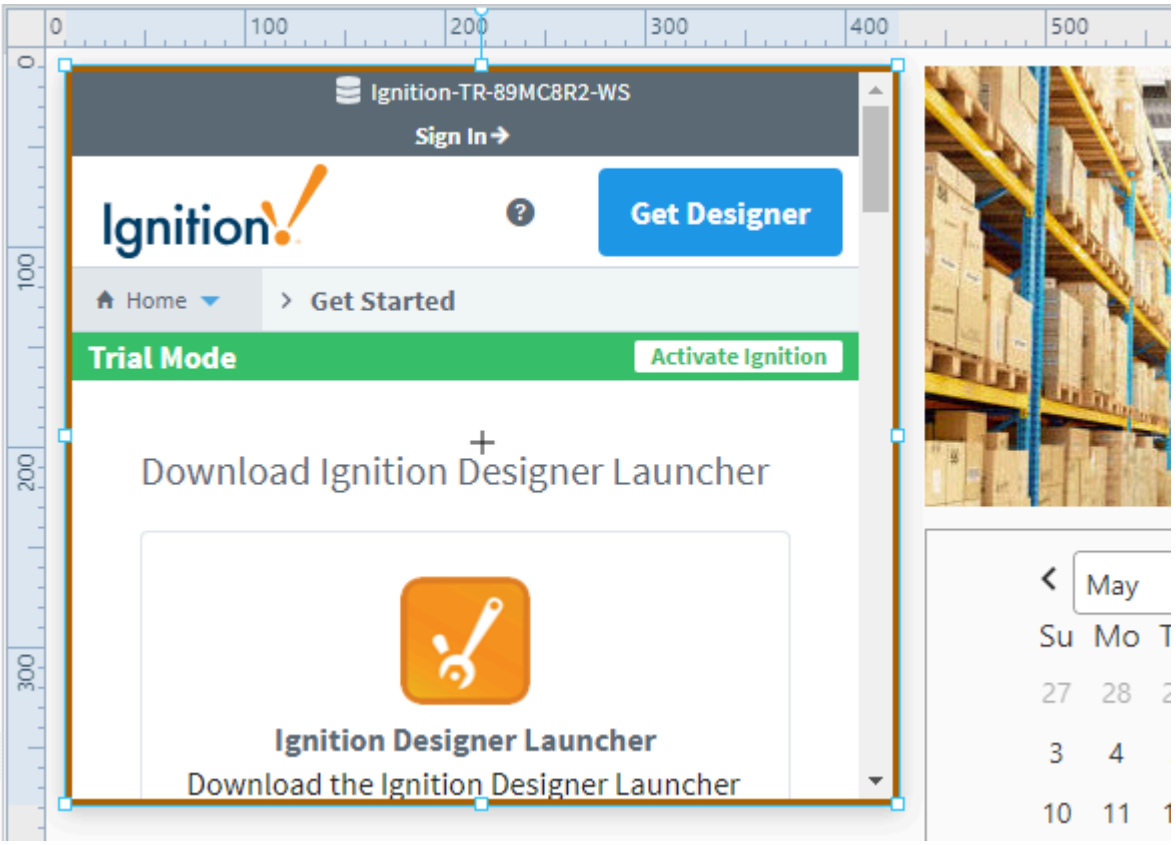
Property	Value
props.source	/system/images/Builtin/Flow/Flow 7.png
props.fit.mode	contain
props.tint.enabled	true
props.tint.color	#FFF00
position.width	150
position.height	115

Example 2



Property	Value	S C
props.source	https://inductiveautomation.com/static/images/logos/inductive-automation-logo.png	N
props.style.borderStyle	solid	b
props.style.borderWidth	1px	b

Perspective - Inline Frame



Component Palette Icon:



Description

The Inline Frame component enables you to display a webpage within the component, allowing another HTML page to be embedded in the view.

Note that many websites will not support rendering if they're inside a frame, such as this component. Website choose to opt in to this via the x-frame-options HTTP header, which all browsers support. The **x-frame-options** header is designed to help prevent a class of web security attacks called Clickjacking.

Thus, if a given page returns a DENY or SAMEORIGIN value for x-frame-options, then a web browser will refuse to render the content in the iFrame on your Perspective page.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
src	The source URL of the webpage you want to embed in this frame.	value: string
allowFullScreen	Whether or not to allow embedded webpage to display full screen. Default is false.	value: boolean
referrerPolicy	Referrer Policy is used to determine what information is sent along with the requests. The referral value is stripped when going from a page using HTTPS to a page using the HTTP protocol. This is because this is the default setting for the Referrer Policy if nothing is specified. Technically, this is “no-referrer-when-downgrade,” which means it will strip the referral when downgrading to an insecure request like switching from HTTPS to HTTP. You don’t have to use the default setting, though. Options as follows: • no-referrer • no-referrer-when-downgrade • origin • origin-when-cross-origin • unsafe-url	value: string
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
border	Specifies how the border property is set on the webpage inside the Inline Frame. The default is unset. To set a border, use the borderStyle property in Style on this component. Using the borderStyle property on the component overrides the border property.	value: string

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

> **CONSIDERATIONS FOR PRINTING INLINE FRAME CONTENTS**
>
> You can use the **Request Print** action or the `self.requestPrint()` function to print static Inline Frame component contents. However, due to the react-to-print library that is used for both print options, requesting to print any content that needs to load or refresh regularly may result in an incomplete image. For instance, charts in a realtime mode are considered dynamic, so the print could result in a partial image since the content in the iFrame may not be finished loading before the frame is printed.

Example

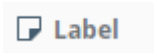


Property	Value
props.src	https://www.youtube.com/embed/hYXUZeLw5ek
props.allowFullScreen	true
props.referrerPolicy	origin
props.style.borderStyle	ridge
props.style.borderWidth	6px
props.style.borderColor	#FF8C00

Perspective - Label



Component Palette Icon:



Description

The Label component displays text and can be customized with a full menu of **style options** for the appearance of text, background, border, color, etc. You can use bindings to display additional information on the Label component.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

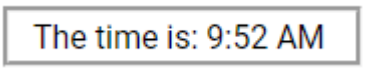
Name	Description	Property Type
text	Text to display. Text can also be entered directly to the label by deep selecting the component, which enables inline editing. Changes are immediately reflected in the text property field.	value: string
alignVertical	Vertical alignment of the text within the component (top, center, or bottom), based on the dimensions of the component. Default is top.	value: string
textStyle	Sets a style for the text within this label. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Example 1



Property	Value
props.text	(Expression Binding) `"The time is : "+dateformat(now(),"h:mm a")`
props.style.borderStyle	groove
props.style.padding	12px

Example 2

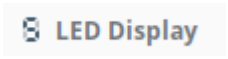


Property	Value
props.text	Lorem ipsum dolor sit amet, consectetur adipiscing elit.
props.textStyle.overflow	hidden
props.textStyle.whiteSpace	nowrap
props.textStyle.textOverflow	ellipsis
props.style.borderStyle	double

Perspective - LED Display



Component Palette Icon:



Description

The LED Display is a stylized numeric and/or alphanumeric label. You can choose between a 7 segment and 14 segment visualization style, and also between nine common number format patterns.

The LED Display component has two pre-configured **variants** for the 7 Segment and 14 Segment styles.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	Value to be displayed.	value: numeric
segmentFormat	Style of each character/digit and the number of segments that compose the character. There are two different visual styles: 7 segment and 14 segment. Default is 14 segment.	value: string
numberFormat	Format of display for numeric characters, including commas, decimal places, percent, etc. There are nine options available from a dropdown list. Default is #,##0.00.	value: string
backgroundColor	Background color of the LED display. Default is #161616. See **Color Selector.**	color
diodeOnColor	Color of LED segments when switched on. Default is #1EC963. See **Color Selector.**	color
diodeOffColor	Color of LED segments when switched off. Generally different from display background color to preserve analog look. See **Color Selector.**	color
locale	Localization code that determines rules for commas, decimals, etc. Default is en-US.	value: string
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

Example 1



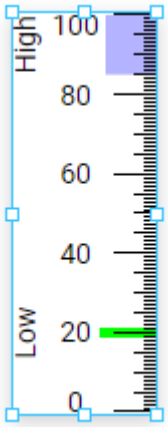
Property	Value
props.value	12.34
props.segmentFormat	7 segment
props.backgroundColor	#D5D5D5
props.diodeOnColor	#0062FF
props.OffColor	#0062FF1A

Example 2

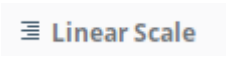


Property	Value	Style Category
props.value	52,345.9	N/A
props.numberFormat	#,##0.00	N/A
props.backgroundColor	#000000	N/A
props.diodeOnColor	#00FF00	N/A
prop.diodeOffColor	#000000	N/A
props.style.borderStyle	groove	border
props.style.padding	2px	margin and padding

Perspective - Linear Scale



Component Palette Icon:



Description

The Linear Scale component displays a series of tick marks and labels that represent a range between a minimum value and a maximum value. It also displays indicators that represent a value or range of values, correctly positioned on the linear scale.

Linear Scale component allows floating point tick marks such as 0.25, 0.5, 0.75, 1.25, etc.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
minValue	The minimum value displayed on the scale.	value: numeric
maxValue	The maximum value displayed on the scale.	value: numeric
majorTicks	Settings for the major tick marks on the scale. See the **majorTicks** properties.	object
minorTicks	Settings for the minor tick marks on the scale. See the **minorTicks** properties.	object
fineTicks	Settings for the fine tick marks on the scale. See the **fineTicks** properties.	object
labels	Displays of the numeric values of major tick marks. Options are as follows: • **angle**: Rotation of the numeric labels. Default is 0. Value is numeric. • **style**: Sets a style for the label. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
indicators	Markers of special significance that can be placed along the scale. Click **here** to see the indicators properties.	object
mirror	Aligns the scale to the opposite side of the component. Default is false.	value: boolean
reverse	Inverts the order of the scale values so min to max is ordered in reverse. Default is false (minimum to max).	value: boolean

Name	Description	Property Type
style	Sets a style for this scale. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Tick Child Properties

The following child properties apply to majorTicks, minorTicks, and fineTicks.

Name	Description	Property Type
span	Distance between each tick mark of this type, in pixels. Default is 20.	value: numeric
length	Length of each tick mark, in pixels. Default is 20.	value: numeric
color	Color for the major ticks. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector.**	color
stroke	Width of each tick mark, in pixels. Default is 1.	value: numeric

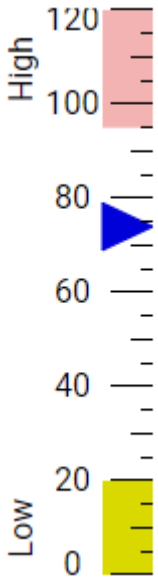
indicators Child Properties

Name	Description	Property Type
value	Numeric value along the scale where the indicator is placed or started.	value: numeric
label	Text to display with the indicator. Default is "High".	value: string
labelColor	Color of the indicator label. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector.**	color
labelAngle	Rotational angle of the indicator label.	value: numeric
color	Color of the indicator or the area making up the indicator. See **Color Selector.**	color
stroke	If indicatorStyle is set to line or wedge, stroke is the width (in pixels) of the indicator.	value: numeric
opacity	Opacity of the indicator. 0 is fully transparent, 1 is fully opaque.	value: numeric
length	Length of the indicator as measured by its x value within the scale. Default is 25.	value: numeric
indicatorStyle	Indicator style can be set to line, wedge, or range. Line is similar to a tick mark. Wedge displays a triangular shape. Range displays a rectangular range along the scale as measured by the property extent. Default is range.	value: string dropdown
extent	If indicatorStyle is set to range, this is the extent along the scale that the indicator is placed. Value must be positive. Default is 15.	value: numeric

Component Events

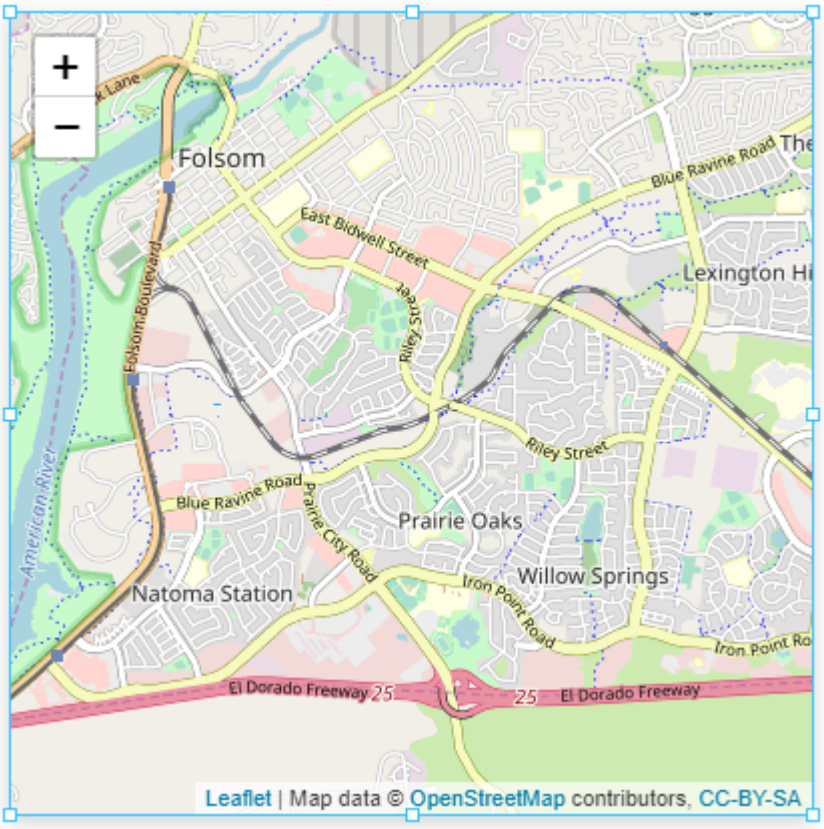
The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example



Property	Value
props.maxValue	120
props.minorTicks.span	10
props.fineTicks.span	5
props.indicators.0.value	95
props.indicators.0.color	#D90000
props.indicators.0.extent	25
props.indicators.1.value	00
props.indicators.1.indicatorStyle	range
props.indicators.1.color	#D9D900
props.indicators.1.extent	20
props.indicators.2.value	74
props.indicators.2.indicatorStyle	wedge
props.indicators.2.color	#0000D9

Perspective - Map



Component Palette Icon:

Map

Description

The Map component provides a mobile-friendly interactive map. Settings can be customized to control the initial view, zoom, mouse interaction, and keyboard interaction.

The Map component is based on the Leaflet open-source JavaScript library for interactive maps. For more information on Leaflet, see **https://leafletjs.com/reference-1.6.0.html**.

User Interaction

Zoom

Depending on the property settings, users can zoom the Map component in several ways:

- Shift and drag the mouse to a rectangular shape.

- Double-click to zoom in and Shift + double-click to zoom out.
- Roll the scroll wheel up to zoom in and down to zoom out.
- Press the + (Plus) key to zoom in and the - (Minus) key to zoom out.

Pan

Depending on the property settings, users can pan across the Map component in multiple ways:

- Use the keyboard arrow keys to pan left, right, up, down.
- Use the mouse to click and drag the map.

Popups

Depending on property settings, users may see the following popup actions.

- Popups close when they click on the map.
- Popups close when they use the ESC key.
- Popups open as they scroll past certain areas of the map.

Properties

Most properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props category of properties. The other categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
init	Map initial state when loaded. Options include: • **center**: Sets the latitude and longitude for the initial state of the map. ◦ **lat**: Latitude value for the map. Value is numeric. ◦ **lng**: Longitude value for the map. Value is numeric. • **zoom**: Initial map zoom level. Percentage value from 1 to 100. Default is 13. Value is numeric.	object
location	Map location. • **enabled**: Enables geolocation. Default is false (disabled). Value is boolean. • **showHeadingIndicator**: Enables the heading indicator. Default is true. Value is boolean.	object
zoom	Zoom properties for the map. See **zoom** properties.	object
customControls	Custom map controls displaying views that are shown in the corner of the map. See **customControls** properties.	object
attribution	Enables an attribution control on the map. Default is true.	array
closePopupsOnClick	When set to true, popups will close when a user clicks anywhere else on the map. Default is true.	value: boolean
trackResize	Enables the map to automatically handle browser window resizing. Default is false.	value: boolean
keyboardNav	Enables navigation of the map with keyboard arrow keys and with the + (Plus) and - (Minus) keys. The + key zooms in; the - key zooms out. Default is true.	value: boolean
keyboardPanDelta	The number of pixels to pan when a keyboard arrow key is pressed. Default is 80.	value: numeric
dragging	Enables the map to be dragged with a mouse/touch. Default is true.	value: boolean

Name	Description	Property Type
maxBounds	When this option is set, the map restricts the view to the given geographical bounds, bouncing the user back if the user tries to pan outside the view. • **corner1** and **corner2**: An object with two elements, each element represents an opposing corner of the rectangle. Each corner contains the following properties: ◦ **lat**: Latitudinal coordinate. Value is numeric. ◦ **lng**: Longitude coordinate. Value is numeric.	object
fadeAnimation	Enables the fade animation. Default is true.	value: boolean
layers	Settings for the map UI, vector, and raster layers that can be displayed on top of the map. • **raster**: Map raster layers. Click to see the **layers.raster** properties. • **vector**: Map vector layers. Allows the map to draw arbitrary shapes at specified coordinates. Click to see the **layers.vector** properties. • **ui**: Map user interface (UI) layers. An array of popup layer configurations for this map. Includes popup location, width, height, and controls for behavior. Value is a string dropdown. Click to see the **layers.ui** properties. • **other**: Other map layers. Click to see the **other.geoJSON** properties. • **hideViewMarkersOnZoom**: Hides view markers while they are being repositioned while zooming in or out. Default value is `true`. Value is boolean.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

zoom Child Properties

Name	Description	Property Type
controls	Whether zoom controls are added to the map. Default is true.	value: boolean
delta	Controls how much the map's zoom level will change after a zoomIn(), zoomOut(), pressing `+` or `-` on the keyboard, or using the zoom controls. Values smaller than 1 allow for greater granularity. Be aware that some combinations of fractional delta and snap values (e.g. delta less than or equal to 0.5 and snap set to 1) may cause unresponsive zoom controls due to rounding.	value: numeric
onBoxZoom	Enables the map to be zoomed to a rectangular area defined by pressing the Shift key while dragging the mouse. Default is true.	value: boolean
onDoubleClick	Enables the map to be zoomed in by double-clicking on it and zoomed out by pressing the Shift key while double-clicking. Default is true.	value: boolean
onScrollWheel	Enables the map to be zoomed in and out using the mouse scroll wheel. Default is true.	value: boolean
snap	Forces the map's zoom level to be a multiple of this value, particularly applicable after fitBounds() or a pinch-zoom. Be aware that some combinations of fractional delta and snap values (e.g. delta less than or equal to 0.5 and snap set to 1) may cause unresponsive zoom controls due to rounding.	value: numeric
max	Maximum zoom level of the map. Default is null.	value: numeric
min	Minimum zoom level of the map. Default is null.	value: numeric
animation	Animation settings for the map.	object

Name	Description	Property Type
	• **enabled**: Whether the map zoom animation is enabled. Value is boolean. • **threshold**: Won't animate zoom if the zoom difference exceeds this value. Default is 4. Value is numeric.	

customControls Child Properties

Property	Description	Property Type
path	Path to the view that will be used as the display for the control. If a path is present in the path property field, an Open View  icon will appear and will navigate directly to the view when clicked.	value: string
params	Parameters to be passed to the view within the control. Names in this object must match input parameters defined on the view.	object
position	Corner of the map where the control will be anchored. Options are: bottom-right, top-right, or bottom left.	value: string
enabled	Whether or not the custom control is enabled.	value: boolean

layers Child Properties

raster Properties

Name	Description	Property Type
tile	Settings enable you to load and display tile layers on the map. The tile provider can be customized to change the appearance of the map. See **https://leaflet-extras.github.io/leaflet-providers/preview/** for some possibilities. • **url**: URL can be either a URL template if using a standard tile service or a Web Map Services URL. Default is `https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png`. In cases where you have the tile files yourself, they need to be hosted on a web server of some sort. Alternatively **WebDev Mounted Folders** can be used to host the tiles. Value is string. • **urlTemplateParams**: An object of parameters to use with a URL template string, if the `url` property is a template string (i.e, contains `{z}/{x}/{y}`). • **WMS**: Web Map Service tile layer provider configurations. Used only if the `url` property is not a template string. See: **https://en.wikipedia.org/wiki/Web_Map_Service**. ◦ **layers**: Comma-separated list of WMS layers to display. (Required). Value is string. ◦ **styles**: Comma-separated list of WMS styles. Value is string. ◦ **format**: WMS image format (use 'image/png' for layers with transparency). Value is string. ◦ **transparent**: If enabled, the WMS service will return images with transparency. Default is false. Value is boolean. ◦ **version**: Version of the WMS service to use. Value is string. ◦ **uppercase**: If enabled, WMS request parameters keys will be uppercase. Default is false. Value is boolean. • **options**: Options for standard tile layer creation. See the **tile.options Properties** table below for options properties.	object
image	Settings for images displayed over specific bounds of the map. • **source**: Image URL. Value is string.	array

Name	Description	Property Type
	• **bounds**: Rectangle bounds. ◦ **corner1** and **corner2**: An object with two elements, each element represents an opposing corner of the rectangle. Each corner contains the following properties: • **options**: Options for the image overlay. ◦ **opacity**: The opacity of the image overlay. Value is numeric. ◦ **alt**: Text for the alt attribute of the image (useful for accessibility). Value: string. ◦ **crossOrigin**: Whether the crossOrigin attribute will be added to the image. If a string is provided, the image will have its crossOrigin attribute set to the String provided. Value is string. ◦ **errorOverlayUrl**: URL to the overlay image to show in place of the overlay that failed to load. Value is string. ◦ **zIndex**: The explicit z-index of the image layer. Value is numeric.	

tile.options Properties

Name	Description	Property Type
attribution	This tile layer's attribution.	value: string
opacity	Opacity of tiles.	value: numeric
zIndex	The z-index of tiles in the grid.	value: numeric
tileSize	Width and height of tiles in the grid. • **width**: Width of tiles in the grid. Value is numeric. • **height**: Height of tiles in the grid. Value is numeric.	object
update	Tile update options. • **whenZooming**: By default, a smooth animation will update grid layers at every integer zoom level. Setting this to false will update the grid layer only when the smooth animation ends. Default is true. Value is boolean. • **whenIdle**: Load new tiles only when panning ends. True by default on mobile browsers, in order to avoid too many requests and keep smooth navigation. Default is false otherwise, in order to display new tiles during panning. Value is boolean. • **interval**: Tiles will not update more than once every update interval in milliseconds when panning. Value is numeric.	object
zoom	Zoom options. • **max**: Maximum zoom level up to which this layer will be displayed (inclusive). Default is 18. Value is numeric. • **min**: Minimum zoom level down to which this layer will be displayed (inclusive). Default is 0. Value is numeric. • **offset**: Zoom number used in the tile URLs will be offset with this value. Value is numeric.	object

Name	Description	Property Type
	• **reverse**: If set to true, the zoom number used in the tile URLs will be reversed (maxZoom to zoom instead of zoom to maxZoom). Default is false. Value is boolean. • **native**: Native zoom levels. ◦ **max**: Maximum zoom number the tile source has available. If specified, the tiles on all zoom levels higher than maxNativeZoom will be loaded from maxNativeZoom level and auto-scaled. Default is null. Value is numeric. ◦ **min**: Minimum zoom number the tile source has available. If specified, the tiles on all zoom levels lower than minNativeZoom will be loaded from minNativeZoom level and auto-scaled. Default is null. Value is numeric.	
subdomains	Subdomains of the tile service. Passed in the form of an array of strings (where each string is a subdomain name). For example, ['a','b','c']. • **errorTileUrl**: URL to the tile image to show in place of the tile that failed to load. Value is string. • **tms**: If true, inverses Y axis numbering for tiles (turn this on for Tile Map Service services). Default is false. Value is boolean. • **detectRetina**: If true and user is on a retina display, it will request four tiles of half the specified size and a bigger zoom level in place of one to utilize the high resolution. Default is false. Value is boolean. • **crossOrigin**: Enables the crossOrigin attribute to be added to the tiles. If a string is provided, all tiles will have their crossOrigin attribute set to the String provided. Default is false. Value is boolean.	array

vector Properties

polygon

An array of polygon layers.

Name	Description	Property Type
polygons	The default shape of polygons consists of the following: • **name**: A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers. • **properties**: Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers. • **points**: An array of points that make up this polygon. ◦ **lat**: Latitudinal coordinate. Value is numeric. ◦ **lng**: Longitude coordinate. Value is numeric.	array
smoothFactor	How much to simplify each vector on each zoom level. Higher the number means better performance and smoother look, and the lower the number means more accurate representation.	value: numeric
noClip	Disables polyline clipping.	value: boolean
event	Event settings for the individual polygon layer. • **stopPropagation**: When enabled, this prevents other mouse event interactions from happening when this vector layer is clicked. Value is boolean.	object
stroke	Stroke settings for the individual polygon layer. • **enabled**: Whether to draw stroke along the path. Set it to false to disable borders. Value is boolean. • **color**: Stroke color. Value is string. • **weight**: Stroke weight in pixels. Value is numeric. • **opacity**: Stroke opacity (0-1). Value is numeric. • **dashArray**: Stroke dash array. Value is numeric. • **dashOffset**: Defines the distance in the dash pattern to start the dash. Value is numeric.	object

Name	Description	Property Type
	• **lineCap**: A string that defines shape to be used at the end of the stroke. Options are round, butt, or square. Value is string. • **lineJoin**: A string that defines shape to be used at the corners of the stroke. Options are round, arc, bevel, miter, or miter-clip. Value is string.	
fill	Fill settings for the individual polygon layer. • **enabled**: Whether to fill the pattern with color. Value is boolean. • **color**: Fill color. Value is string. • **opacity**: Fill opacity (0-1). Value is numeric. • **rule**: Presentation attribute defining the algorithm to use to determine the inside part of the shape. Options are nonzero or evenodd. Value is string.	object

polyline

An array of polyline layers.

Name	Description	Property Type
polylines	The default shape of polylines consists of the following: • **name**: A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers. • **properties**: Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers. • **points**: An array of points that make up this polyline. ◦ **lat**: Latitudinal coordinate. Value is numeric. ◦ **lng**: Longitude coordinate. Value is numeric.	array
smoothFactor	How much to simplify each polyline on the zoom level.	value: numeric
noClip	Disables polyline clipping.	value: boolean
event	Event settings for the individual polygon layer. • **stopPropagation**: When enabled, this prevents other mouse event interactions from happening when this vector layer is clicked. Value is boolean.	object
stroke	Stroke settings for the individual polygon layer. • **enabled**: Whether to draw stroke along the path. Set it to false to disable borders. Value is boolean. • **color**: Stroke color. Value is string. • **weight**: Stroke weight in pixels. Value is numeric. • **opacity**: Stroke opacity (0-1). Value is numeric. • **dashArray**: Stroke dash array. Value is numeric. • **dashOffset**: Defines the distance in the dash pattern to start the dash. Value is numeric. • **lineCap**: A string that defines shape to be used at the end of the stroke. Options are round, butt, or square. Value is string.	object

Name	Description	Property Type
	• **lineJoin**: A string that defines shape to be used at the corners of the stroke. Options are round, arc, bevel, miter, or miter-clip. Value is string.	
fill	Fill settings for the individual polygon layer. • **enabled**: Whether to fill the pattern with color. Value is boolean. • **color**: Fill color. Value is string. • **opacity**: Fill opacity (0-1). Value is numeric. • **rule**: Presentation attribute defining the algorithm to use to determine the inside part of the shape. Options are nonzero or evenodd. Value is string.	object

rectangle

An array of rectangle layers.

Name	Description	Property Type
rectangles	An array of rectangle bounds each consisting of two corners that create a single rectangle. • **name**: A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers. • **properties**: Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers. • **corner1** and **corner2**: An object with two elements, each element represents an opposing corner of the rectangle. Each corner contains the following properties: ◦ **lat**: Latitudinal coordinate. Value is numeric. ◦ **lng**: Longitude coordinate. Value is numeric.	object
smoothFactor	How much to simplify each rectangle on the zoom level.	value: numeric
noClip	Disables rectangle clipping.	value: boolean
event	Event settings for the individual rectangle layer. • **stopPropagation**: When enabled, this prevents other mouse event interactions from happening when this vector layer is clicked. Value is boolean.	object
stroke	Stroke settings. • **enabled**: Whether to draw stroke along the path. Set it to false to disable borders. Value is boolean. • **color**: Stroke color. Value is string. • **weight**: Stroke weight in pixels. Value is numeric. • **opacity**: Stroke opacity (0-1). Value is numeric. • **dashArray**: Stroke dash array. Value is numeric.	object

Name	Description	Property Type
	• **dashOffset**: Defines the distance in the dash pattern to start the dash. Value is numeric. • **lineCap**: A string that defines shape to be used at the end of the stroke. Options are round, butt, or square. Value is string. • **lineJoin**: A string that defines shape to be used at the corners of the stroke. Options are round, arc, bevel, miter, or miter-clip. Value is string.	
fill	Fill settings for the individual rectangle layer. • **enabled**: Whether to fill the pattern with color. Value is boolean. • **color**: Fill color. Value is string. • **opacity**: Fill opacity (0-1). Value is numeric. • **rule**: Presentation attribute defining the algorithm to use to determine the inside part of the shape. Options are nonzero or evenodd. Value is string.	object

circle

An array of circle layers.

Name	Description	Property Type
circles	An array of circles each consisting of center point corners that creates a single circle. • **name**: A unique name which is used to distinguish this element from others. This name is provided to any applicable interaction event handlers, such as click handlers. • **properties**: Properties of this element. These are provided to any applicable interaction event handlers, such as click handlers. • **lat**: Latitudinal coordinate. Value is numeric. • **lng**: Longitude coordinate. Value is numeric.	object
radius	Radius of the circle marker, in pixels.	value: numeric
event	Event settings for the individual circle layer. • **stopPropagation**: When enabled, this prevents other mouse event interactions from happening when this vector layer is clicked. Value is boolean.	object
stroke	Stroke settings for individual circle layer. • **enabled**: Whether to draw stroke along the path. Set it to false to disable borders. Value is boolean. • **color**: Stroke color. Value is string. • **weight**: Stroke weight in pixels. Value is numeric. • **opacity**: Stroke opacity. Value is numeric. • **dashArray**: Stroke dash array. Value is numeric. • **dashOffset**: Defines the distance in the dash pattern to start the dash. Value is numeric. • **lineCap**: A string that defines shape to be used at the end of the stroke. Options are round, butt, or square. Value is string. • **lineJoin**: A string that defines shape to be used at the corners of the stroke. Options are round, arc, bevel,	object

Name	Description	Property Type
	miter, or miter-clip. Value is string.	
fill	Fill settings for the individual circle layer. • **enabled**: Whether to fill the pattern with color. Value is boolean. • **color**: Fill color. Value is string. • **opacity**: Fill opacity (0-1). Value is numeric. • **rule**: Presentation attribute defining the algorithm to use to determine the inside part of the shape. Options are nonzero or evenodd. Value is string.	object

ui Properties

marker

Map marker layers.

Name	Description	Property Type
name	A unique name that is used to distinguish this marker from another.	value: string
enabled	Determines if the marker is shown on the map. Default value is true.	value: boolean
lat	Latitudinal coordinate.	value: numeric
lng	Longitudinal coordinate.	value: numeric
opacity	Marker opacity.	value: numeric
icon	Marker icon configuration. • **path**: Icon path. Otherwise uses default icon. Value is string. • **color**: Fill color for the icon. Value is string. • **size**: Size settings. ◦ **width**: Width in pixels. Value is numeric. ◦ **height**: Height in pixels. Value is numeric. • **style**: Style for the icon. Full menu of style options is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a style class. • **rotate**: Rotation degrees of the specified map marker. Value is string.	object
tooltip	This marker's tooltip configuration, if tooltip is enabled.	object
popup	Popup configuration for this marker.	

popup

Array of popup layer configurations for this map.

Name	Description	Property Type
enabled	Determines if the popup is shown on the map. Default value is false.	value: boolean
lat	Latitudinal coordinate.	value: numeric
lng	Longitudinal coordinate.	value: numeric
content	• **content**: The tooltip content to display. ◦ **text**: Text to display. Value is string. ◦ **view**: A view to display as tooltip content. If configured, overrides the text property. ▪ **path**: Path of view to display. Value is string. ▪ **params**: Parameters for the view. If passing parameters into the embedded view, the names here must match the parameters on that view.	object
width	Width of the popup. Min and Max values allowed.	value: numeric
height	Max popup height in pixels.	value: numeric

Name	Description	Property Type
pan	Popup pan configuration. • **closeButton**: Controls the presence of a close button in the popup. Value is boolean. • **autoClose**: Set it to false if you want to override the default behavior of the popup closing when another popup is opened. Value is boolean. • **closeOnEscapeKey**: Set it to false if you want to override the default behavior of the ESC key for closing the popup. Value is boolean. • **closeOnClick**: Set it if you want to override the default behavior of the popup closing when a user clicks on the map. Value is boolean.	

view

Array of view layer configurations for this map.

Name	Description	Property Type
path	Path to the view	value: string
params	Params to pass to the view	object
lat	Latitudinal coordinate.	value: numeric
lng	Longitudinal coordinate.	value: numeric
transparentBackground	If enabled, disables the default background color. Default is false.	value: boolean
shadow	If enabled, apply a box shadow around the view. Default is true.	value: boolean

other.geoJSON Properties

GeoJSON objects to include as a feature layer. Each element should be a geoJSON FeatureCollection. This property is intended to be modified by an HTTP binding to a URL containing the desired geoJSON data, and a map transform to process the results.

For more information on the geoJSON format, see **http://geojson.org/**.

It is not the intent that geoJSON shapes are manually added to the component. In addition, you can use following site to create a basic geoJSON layer: **https://geojson.io/**.

See **Perspective Map - Adding geoJSON Shapes** for more information.

Property	Description	Property Type
styleOptions	Style options for a FeatureCollection layer. Additional styleOptions objects can be added to individual features within the FeatureCollection. • **stroke**: Whether to draw stroke along the path. Set it to false to disable borders on polygons or circles. Value is boolean. • **color**: Stroke color. • **weight**: Stroke width in pixels. Value is numeric. • **opacity**: Stroke opacity. Value is numeric. • **lineCap**: A string that defines the shape to be used at the end of the stroke. Options include butt, round, square, and inherit. Default value is inherit. Value is string. • **lineJoin**: A string that defines shape to be used at the corners of the stroke. Options include miter, round, bevel, and inherit. Default value is inherit. Value is string. • **dashArray**: A string that defines the stroke **dash pattern**. Value is string. • **dashOffset**: A string that defines the **distance into the dash pattern to start the dash**. Value is string. • **fill**: Whether to fill the path with color. Set it to false to disable filling on polygons or circles. Value is boolean. • **fillColor**: Fill color. Defaults to the value of the color property. • **fillOpacity**: Fill opacity. Value is numeric. • **fillRule**: A string that defines **how the inside of a shape** is determined. Value is string. • **interactive**: If false, the layer will not emit mouse events and will act as a part of the underlying map. Value is boolean. • **className**: A custom class name to assign to the layer for additional styling via CSS. Value is string.	object

Map Parameters

Because this component uses the Leaflet library, there are several common objects you'll want to be aware of when using the callable methods on this component. Each of these "objects" is just a Python dictionary with specific keys. Documentation for these objects is provided for convenience.

LatLng

An object representing a point at a certain latitude and longitude. For the purposes of interacting with the map component, this object is typically a Python dictionary.

```
# Example
{"lat":50, "lng":30}
```

Option	Type	Default	Description
lat	Numeric	None	Numerical value representing a latitude value.
lng	Numeric	None	Numerical value representing a longitude value.

PanOptions

A Python dictionary containing keys that can modify the panning behavior on the Map component. The contents of this dictionary are listed below, but the original explanation can be found in **Leaflet's documentation**.

Option	Type	Default	Description
animate	Boolean	false	If true, panning will always be animated if possible. If false, it will not animate panning, either resetting the map view if panning more than a screen away, or just setting a new offset for the map pane (except for panBy which always does the latter).
duration	Numeric	0.25	Duration of animated panning, in seconds.
easeLinearity	Numeric	0.25	The curvature factor of panning animation easing (third parameter of the Cubic Bezier curve). 1.0 means linear animation, and the smaller this number, the more bowed the curve.
noMoveStart	Boolean	false	If true, panning won't fire movestart event on start (used internally for panning inertia).

ZoomOptions

A Python dictionary containing keys that can modify the zooming behavior on the Map. The contents of this dictionary are listed below, but the original explanation can be found in **Leaflet's documentation**.

Option	Type	Default	Description
animate	Boolean	false	If true, panning will always be animated if possible. If false, it will not animate panning, either resetting the map view if panning more than a screen away, or just setting a new offset for the map pane (except for panBy which always does the latter).

FitBounds

A Python dictionary containing keys that can modify the zooming behavior on the Map. The contents of this dictionary are listed below, but the original explanation can be found in **Leaflet's documentation**.

Option	Type	Default	Description
paddingTopLeft	Point	[0,0]	Sets the amount of padding in the top left corner of a map container that shouldn't be accounted for when setting the view to fit bounds. Useful if you have some control overlays on the map like a sidebar and you don't want them to obscure objects you're zooming to.
paddingBottomRight	Point	[0,0]	Similar to paddingTopLeft, except for the bottom right corner of the map.
padding	Point	[0,0]	A setting that allows both the top left and bottom right padding to use the same value.
maxZoom	Numeric	null	The maximum possible zoom to use.

Scripting

See the **Perspective - Map Scripting** page for the full list of scripting functions available for this component.

Example

Additional Layers

You can add any number of layers to the Perspective Map component by adding tiles under **layers.raster.tile**.

You will need to obtain an API key to use most map providers. Many API keys are available for free or with a subscription to a service such as Google Maps Platform. The example below makes use of the **Open Weather Map API**.

Open Weather Map API Endpoint

```
https://tile.openweathermap.org/map/{layer}/{z}/{x}/{y}.png?appid={API key}
```

For this example, we only need to update two parameters in the endpoint URL:

Parameter	Description
`{layer}`	The name of the layer to display, such as clouds_new or temp_new.
`{API key}`	Your unique API key.



layers.raster.tile.0

Default map tile

Property	Value
url	`https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png`
options	• opacity: 1 • zIndex: 0

layers.raster.tile.1

Clouds map tile

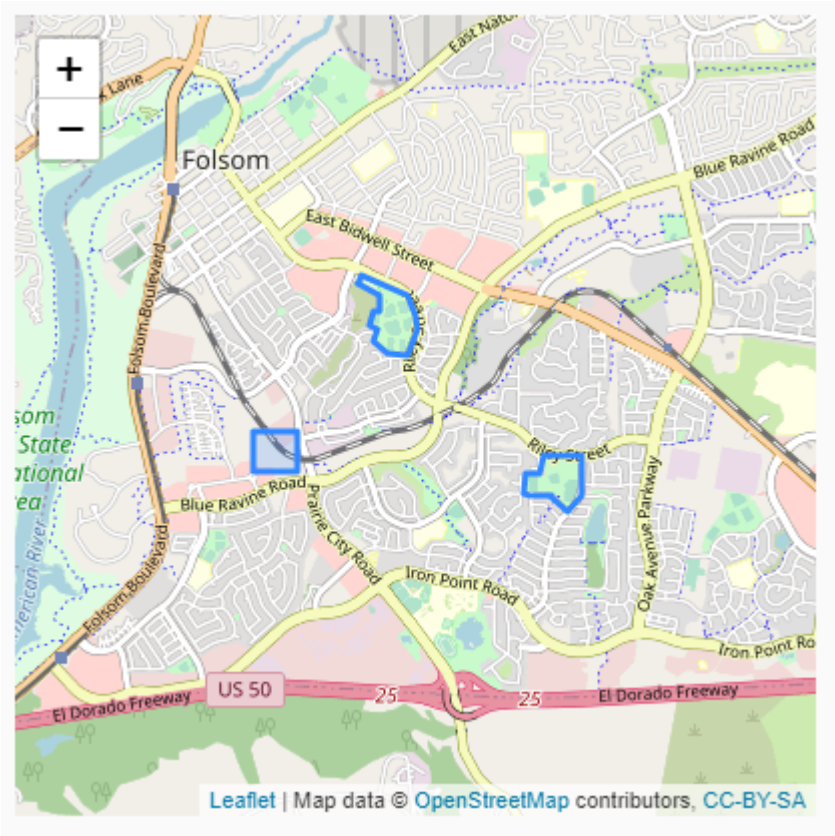
Property	Value
url	`https://tile.openweathermap.org/map/clouds_new/{z}/{x}/{y}.png?appid=YOUR_API_KEY`
options	• opacity: 1 • zIndex: 1

layers.raster.tile.2

Temperature map tile

Property	Value
url	`https://tile.openweathermap.org/map/temp_new/{z}/{x}/{y}.png?appid=YOUR_API_KEY`
options	• opacity: 0.6 • zIndex: 2

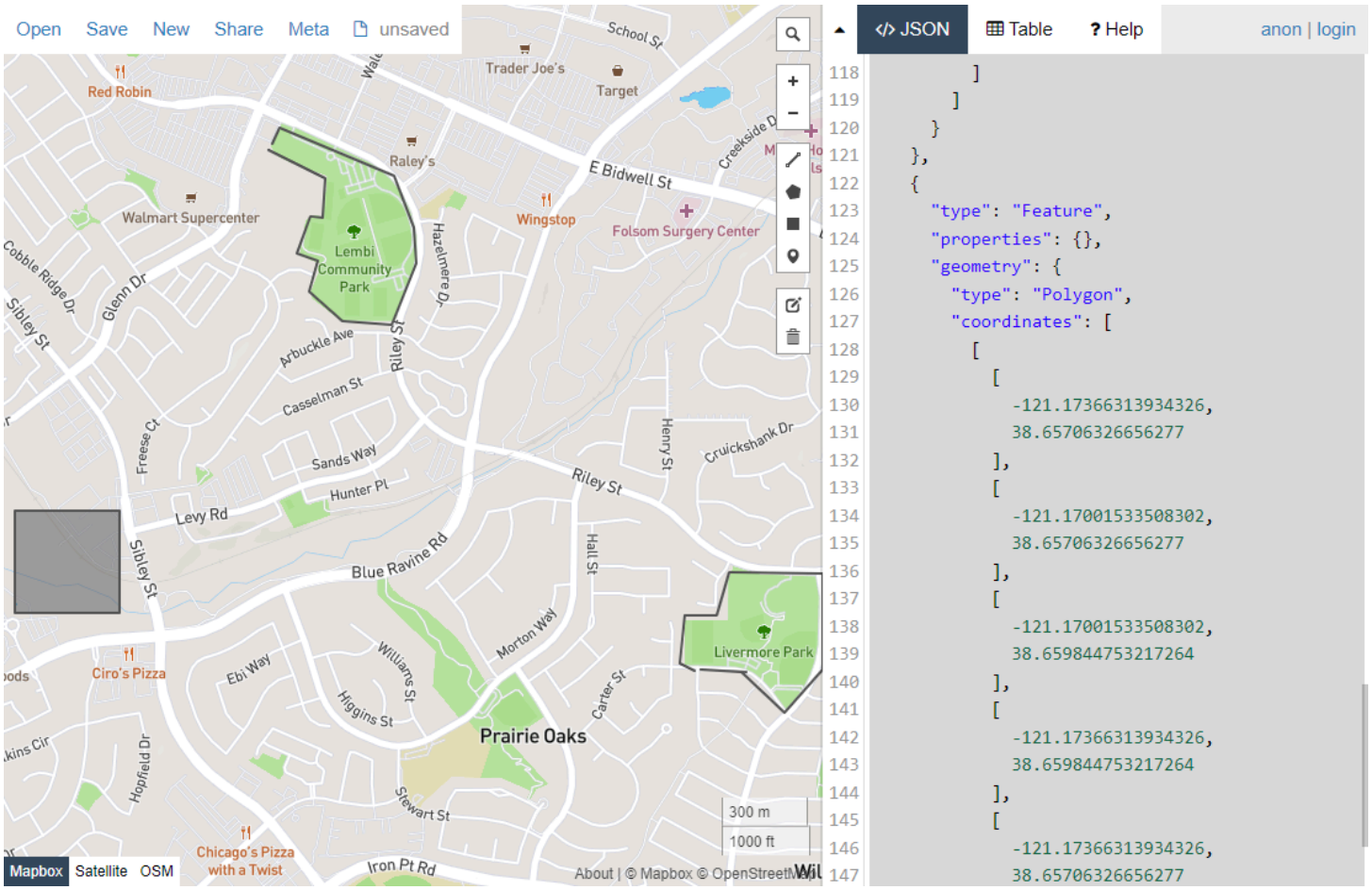
Perspective - Adding GeoJSON Shapes



In this example we'll demonstrate how to add **geoJSON** features to the Map component.

Acquiring GeoJSON Data

The first step involves acquiring some data. Instead of writing this by hand we can use **https://geojson.io/** to quickly draw a few features. In a real world scenario this data would likely be provided by an external endpoint, or harvested from something like data.gov and hosted internally. For this example we're going to generate the geoJSON data, then create a Web Dev resource to store the data.



The code block below contains the data we'll use for this example.

Example GeoJSON Data

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [
            -121.14997386932373,
            38.655488159953
          ],
          [
            -121.15048885345459,
            38.655689239321134
          ],
          [
            -121.15031719207764,
            38.65699624145518
          ],
          [
            -121.14920139312744,
            38.65699624145518
          ],
          [
            -121.14877223968504,
            38.65816917178303
          ],
          [
            -121.14551067352294,
            38.658135659754485
          ],
          [
            -121.14551067352294,
            38.655521673220235
          ],
          [
            -121.14684104919432,
            38.65434869953815
          ],
          [
            -121.14834308624268,
            38.6554546466701
          ],
          [
            -121.14975929260254,
            38.655521673220235
```

```
          ],
          [
            -121.14817142486571,
            38.6554546466701
          ]
        ]
      }
    },
    {
      "type": "Feature",
      "properties": {

      },
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [
            -121.16469383239745,
            38.66983040550309
          ],
          [
            -121.16443634033203,
            38.670265989507826
          ],
          [
            -121.16044521331787,
            38.66895922954322
          ],
          [
            -121.15975856781006,
            38.66755192290758
          ],
          [
            -121.15967273712158,
            38.666580195039295
          ],
          [
            -121.16053104400635,
            38.66490477119321
          ],
          [
            -121.16224765777588,
            38.665005297729465
          ],
          [
            -121.16340637207033,
            38.66580950493899
          ],
          [
```

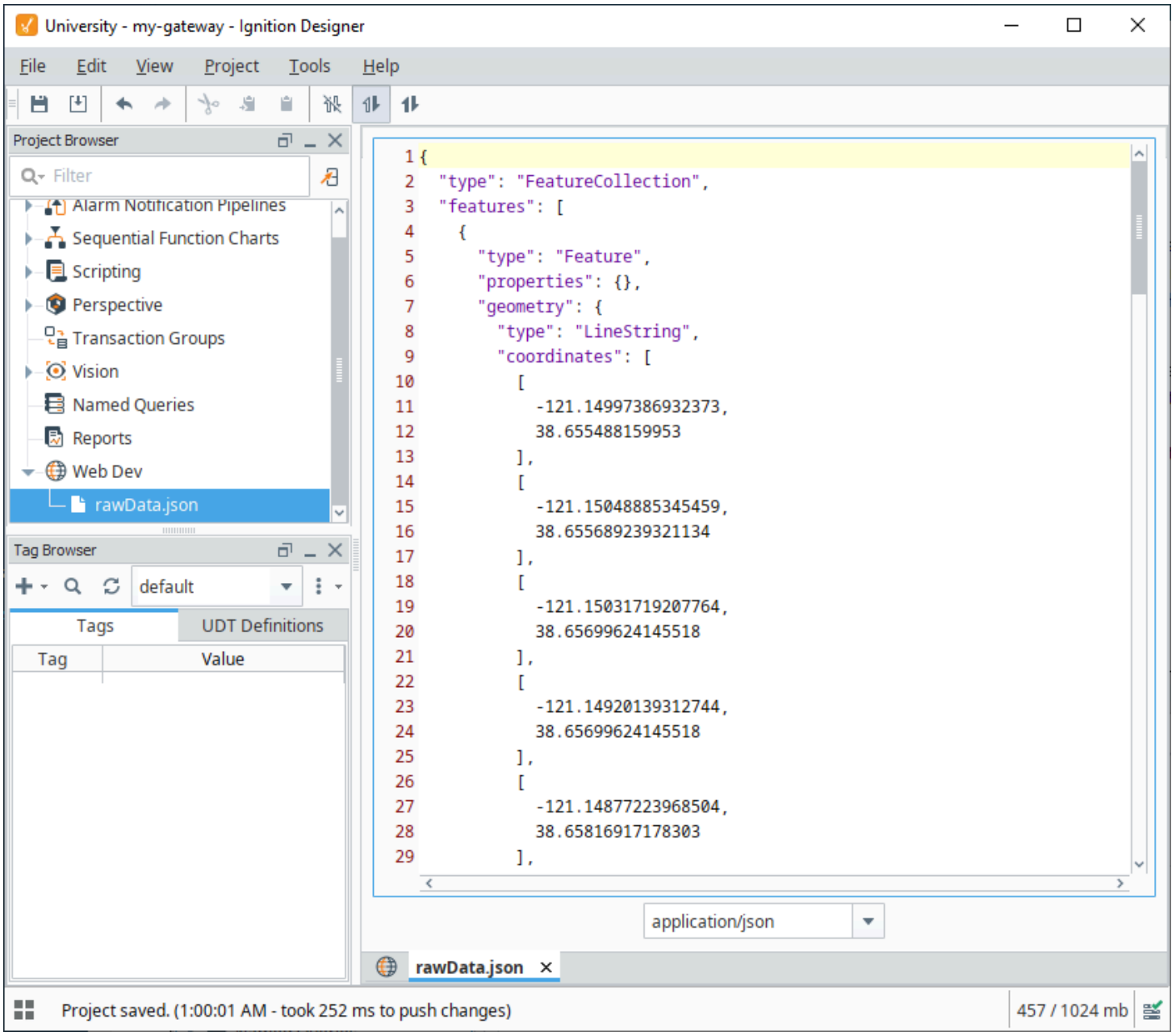
```
            -121.16310596466064,
            38.66664721117775
          ],
          [
            -121.16383552551268,
            38.66684825921672
          ],
          [
            -121.16336345672606,
            38.66785349094385
          ],
          [
            -121.16293430328369,
            38.66781998344699
          ],
          [
            -121.16284847259521,
            38.66889221556877
          ],
          [
            -121.16482257843018,
            38.669662872487926
          ]
        ]
      }
    },
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [
              -121.17366313934326,
              38.65706326656277
            ],
            [
              -121.17001533508302,
              38.65706326656277
            ],
            [
              -121.17001533508302,
              38.659844753217264
            ],
            [
              -121.17366313934326,
              38.659844753217264
            ],
```

```
            [
              -121.17366313934326,
              38.65706326656277
            ]
          ]
        ]
      }
    }
  ]
}
```

Store the Data

Next we can store the data as a **Web Dev** resource. If your data is being provided by an existing endpoint, then skip this section.

1. In the Designer's Project Browser, navigate to the Web Dev section and create a new **Text Resource**. Name this resource **rawData.json**.
2. At the bottom of the workspace, change the dropdown to **application/json**.
3. Paste your geoJSON data into the text resource.
4. Save the Project.



Pass the Data to the Map

Now that we have some data we can hand it off to our map component. We'll use a HTTP binding.

1. In the Designer, navigate to, or create, a Perspective view.

2. On the view, create a Map component.

3. With the Map selected, locate the **props.layers.other.geoJSON** property. Click the **Add Array Element** button, which adds a "0" member.

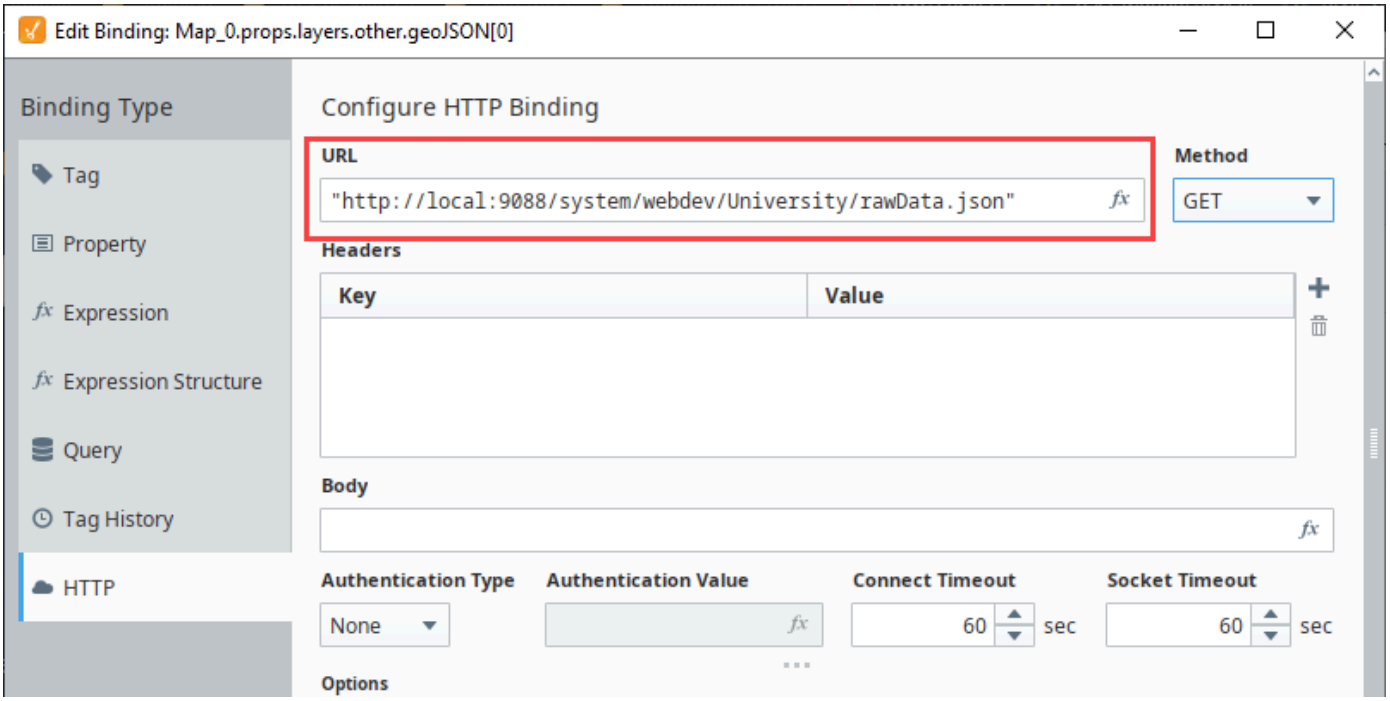
4. Configure a binding on the "0" member.



5. Select **HTTP** as the binding type.

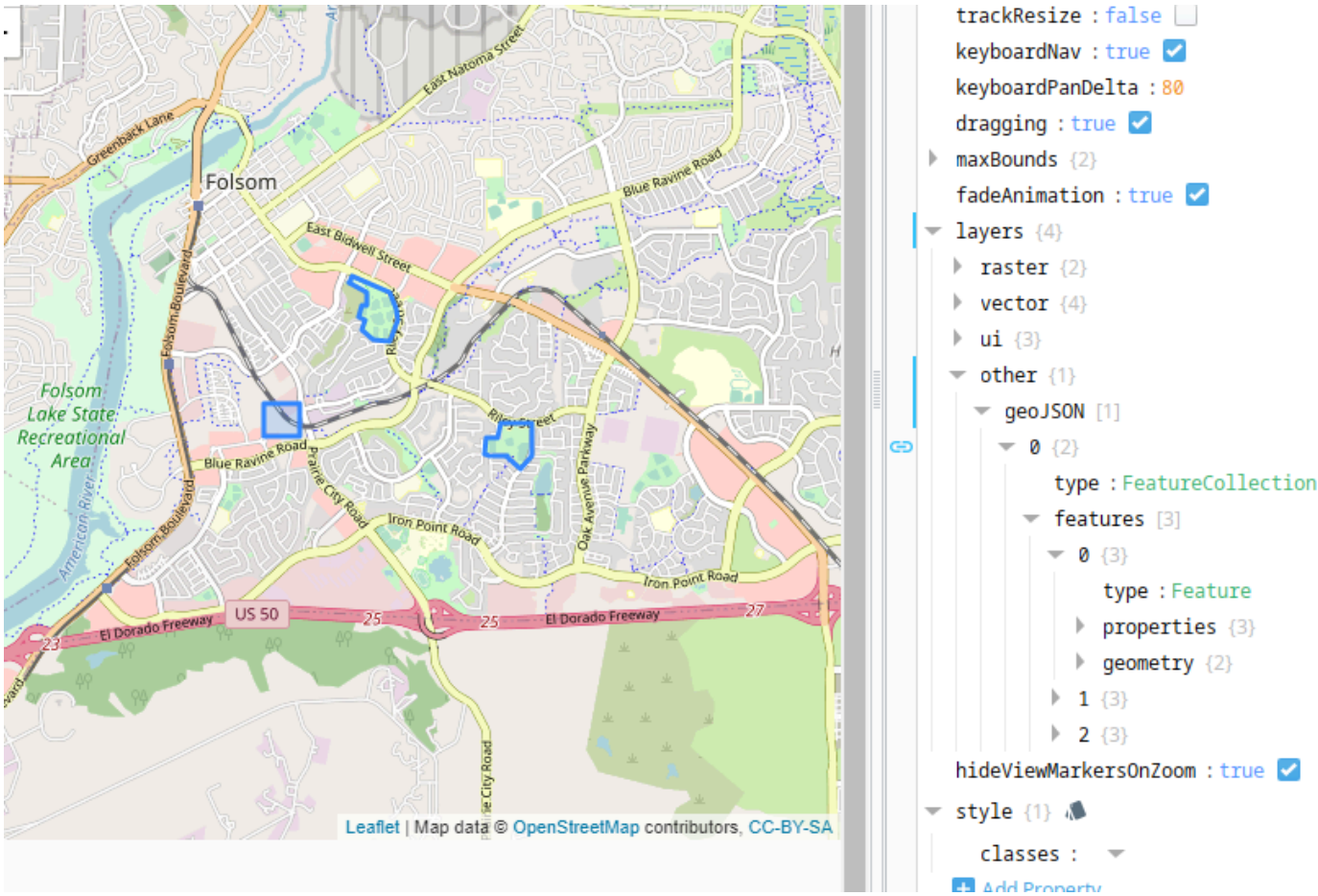
6. For the URL property, enter the endpoint housing your data. If you're using a Web Dev resource as demonstrated earlier on this page then you can **copy the mounted path** from the Project Browser, but make sure to use a full URL by adding in "http://" or "https://".

7. After pasting the URL, place quotation marks on both sides of the URL, as the field expects a valid expression.



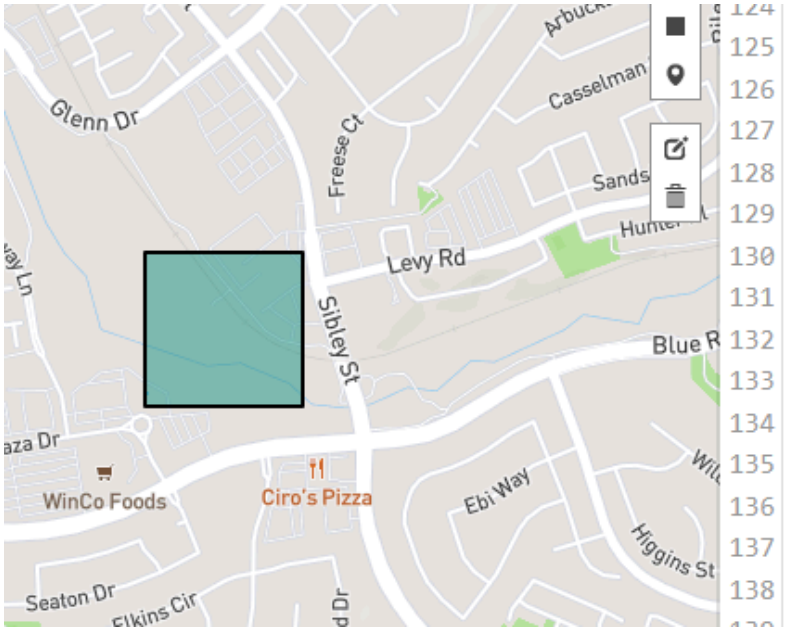
8. Click **OK** to close the binding window.

Now we have some features on our map that are driven by our geoJSON data.



Modify the Incoming Data

Our Map is now showing features. In a real world scenario you may need to modify the incoming data so that it is formatted in a way that the map component expects. For example, the geojson.io tool allows you to define colors for the strokes and fill on each feature. However, it defines these features under a "properties" object, which our Map component cannot utilize since it is expecting any styling attributes to be listed under a "styleOptions" object.




```
},
{
  "type": "Feature",
  "properties": {
    "fill-opacity": 0.5,
    "stroke": "#000000",
    "fill": "#189183"
  },
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [
          -121.17366313934326,
```

For this next example, we will demonstrate how to modify incoming data with a script transform. This example will focus on styling attributes, but could be expanded to account for other arbitrary geoJSON property data.

Update our GeoJSON Data

The code block below contains the new geoJSON data this example will use. Copy the contents of the code block, and paste them into our Web Dev text resource. Save the project after pasting.

Updated GeoJSON Data

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "stroke": "#c7170a"
      },
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [
            -121.14997386932373,
            38.655488159953
          ],
          [
            -121.15048885345459,
            38.655689239321134
          ],
          [
            -121.15031719207764,
            38.65699624145518
          ],
          [
            -121.14920139312744,
            38.65699624145518
          ],
          [
            -121.14877223968504,
            38.65816917178303
          ],
          [
            -121.14551067352294,
            38.658135659754485
          ],
          [
            -121.14551067352294,
            38.655521673220235
          ],
          [
            -121.14684104919432,
            38.65434869953815
          ],
          [
            -121.14834308624268,
            38.6554546466701
          ],
          [
```

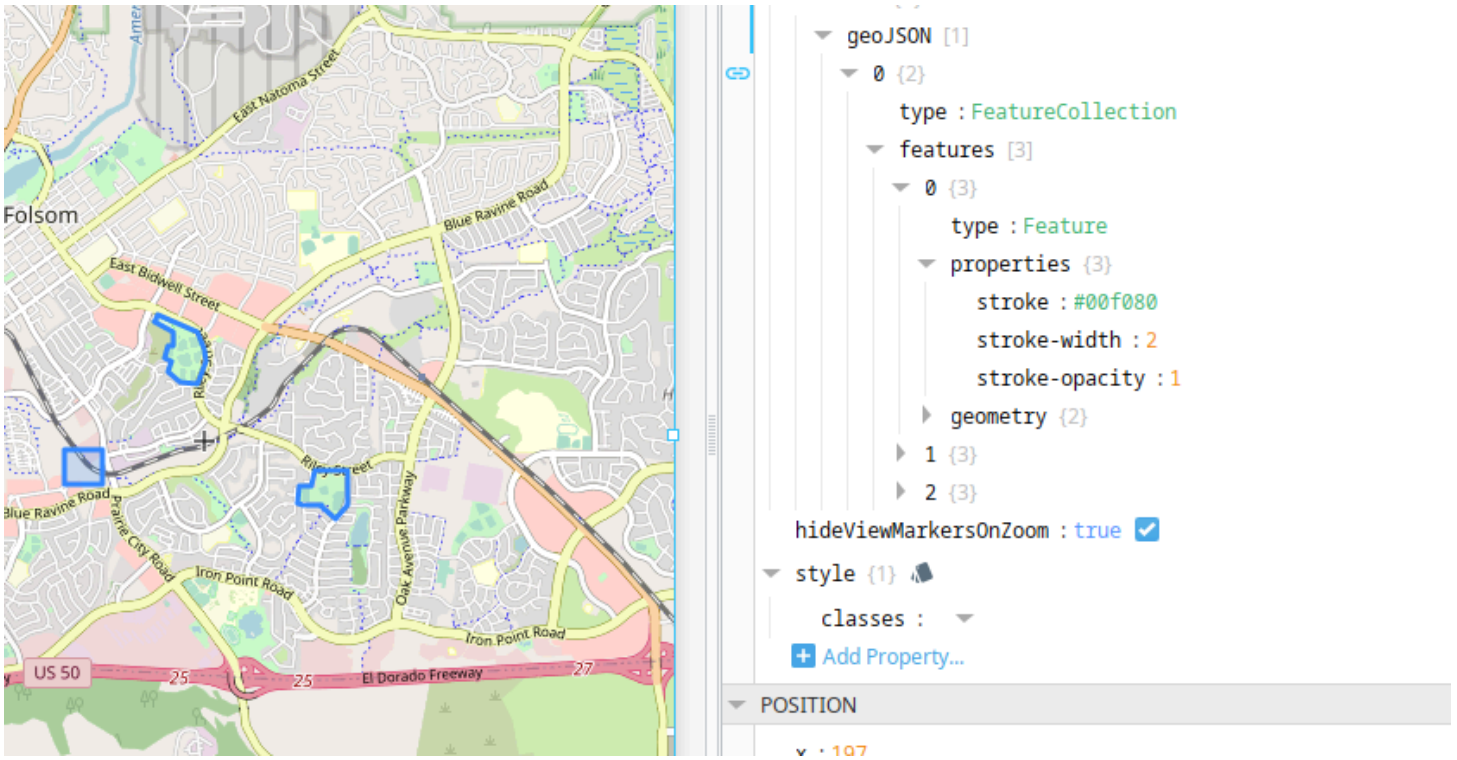
```
            -121.14975929260254,
            38.6555216732202 35
          ],
          [
            -121.14817142486571,
            38.6554546466701
          ]
        ]
      }
    },
    {
      "type": "Feature",
      "properties": {
        "stroke": "#0f3df5"
      },
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [
            -121.16469383239745,
            38.66983040550309
          ],
          [
            -121.16443634033203,
            38.670265989507826
          ],
          [
            -121.16044521331787,
            38.66895922954322
          ],
          [
            -121.15975856781006,
            38.66755192290758
          ],
          [
            -121.15967273712158,
            38.666580195039295
          ],
          [
            -121.16053104400635,
            38.66490477119321
          ],
          [
            -121.16224765777588,
            38.665005297729465
          ],
          [
            -121.16340637207033,
            38.66580950493899
```

```
        ],
        [
          -121.16310596466064,
          38.66664721117775
        ],
        [
          -121.16383552551268,
          38.66684825921672
        ],
        [
          -121.16336345672606,
          38.66785349094385
        ],
        [
          -121.16293430328369,
          38.66781998344699
        ],
        [
          -121.16284847259521,
          38.66889221556877
        ],
        [
          -121.16482257843018,
          38.669662872487926
        ]
      ]
    }
  },
  {
    "type": "Feature",
    "properties": {
      "fill-opacity": 0.5,
      "stroke": "#000000",
      "fill": "#189183"
    },
    "geometry": {
      "type": "Polygon",
      "coordinates": [
        [
          [
            -121.17366313934326,
            38.65706326656277
          ],
          [
            -121.17001533508302,
            38.65706326656277
          ],
          [
            -121.17001533508302,
```

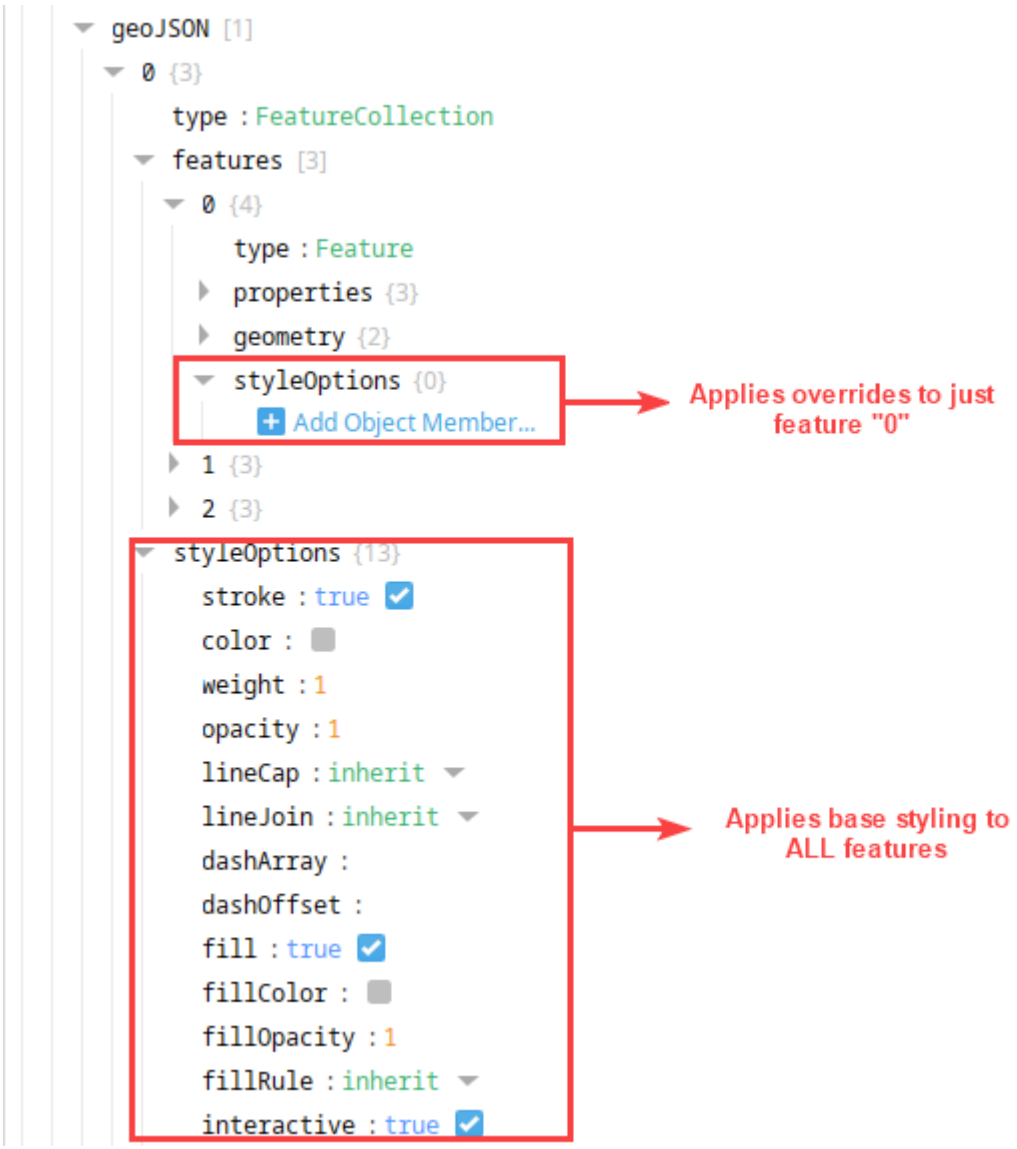
```
            38.659844753217264
          ],
          [
            -121.17366313934326,
            38.659844753217264
          ],
          [
            -121.17366313934326,
            38.65706326656277
          ]
        ]
      ]
    }
  ]
}
```

Update our Binding

Now that our map is using the updated data, you should see that the "properties" object under each feature contains some styling properties. However, they are being ignored. This is because we need to add a "styleOptions" object in each feature, and relocate the styling configurations so they're under the new object.

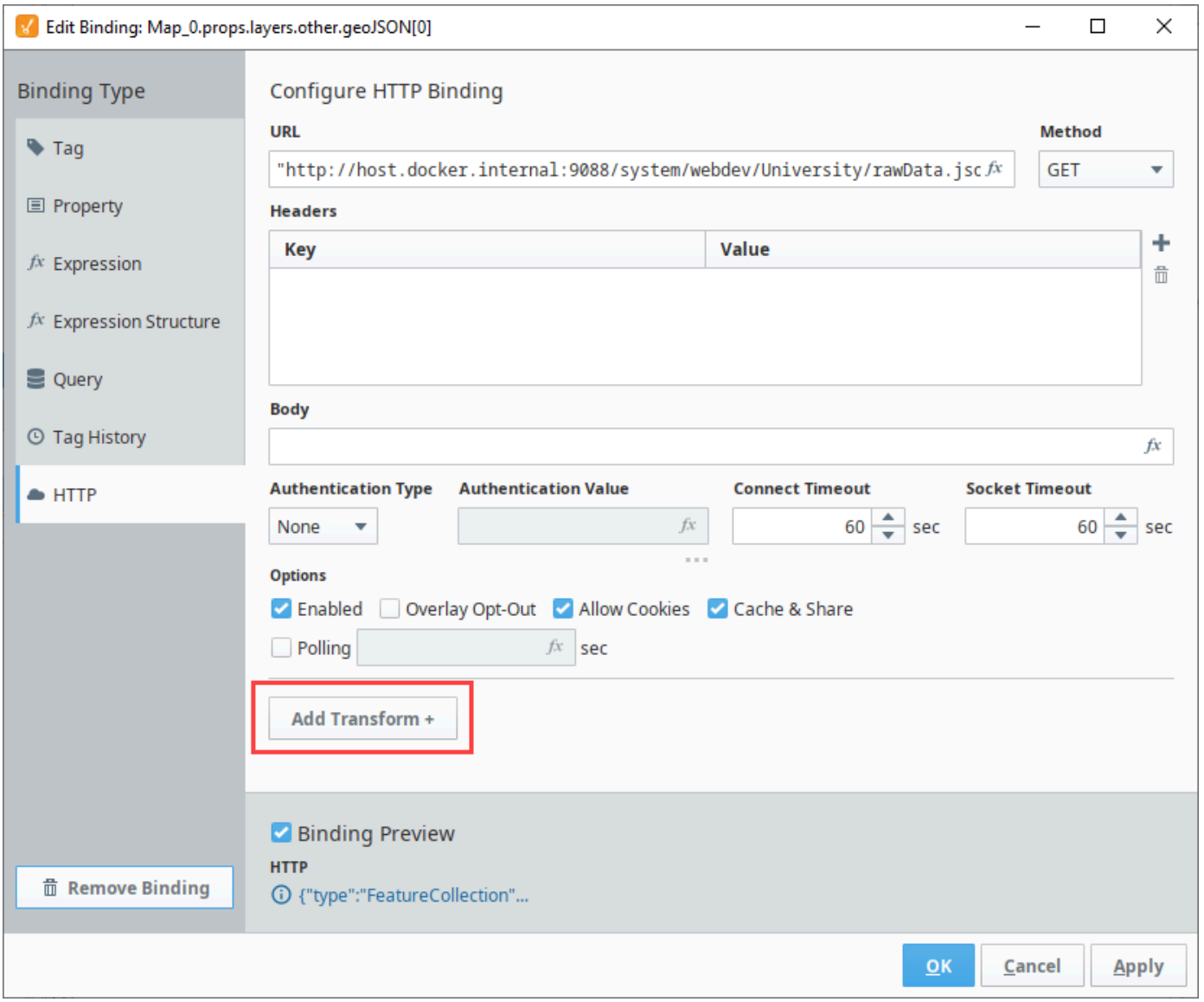


In addition, we should also add a shared styleOptions object as a peer to all of our features. This allows each feature to only override the style settings they need to. In other words, we want something like this:

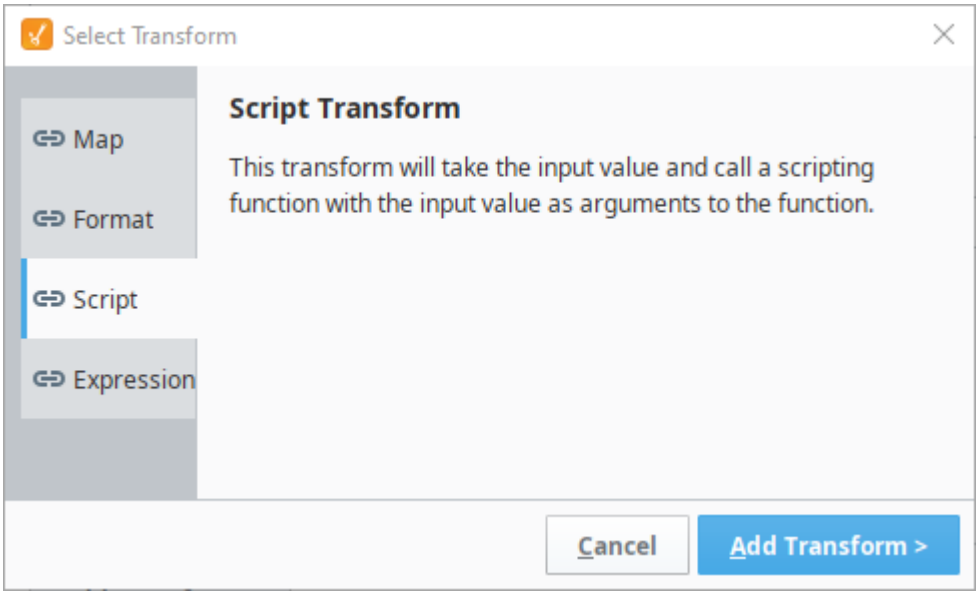


We can do this automatically with a Script Transform.

1. Select the Map component.
2. Locate the binding on **props.layers.other.geoJSON.0**, and click the link icon to open the binding window.
3. Click the **Add Transform** button.



4. Select "Script" and click **Add Transform >**.



5. Locate the first line of editable code. Replace that with the code below.

```
# Define a base style for all features, and add them as an object at the root
of our results.
# The keys below are based upon the expected keys that the Map component
supports.
value["styleOptions"] = {
    "stroke": True,
    "color": "",
    "weight": 2,
    "opacity": 1,
    "lineCap": "inherit",
    "lineJoin": "inherit",
    "dashArray": "",
    "dashOffset": "",
    "fill": False,
    "fillColor": "",
    "fillOpacity": 1,
    "fillRule": "inherit",
    "interactive": True
    }

#Here we iterate all of the features within our data.
for i in value["features"]:

    # Inside each feature, we can examine each key in "properties", and add
custom styling
    i["styleOptions"] = {}
    if i["properties"].has_key("stroke"):
        i["styleOptions"]["color"] = i["properties"]["stroke"]

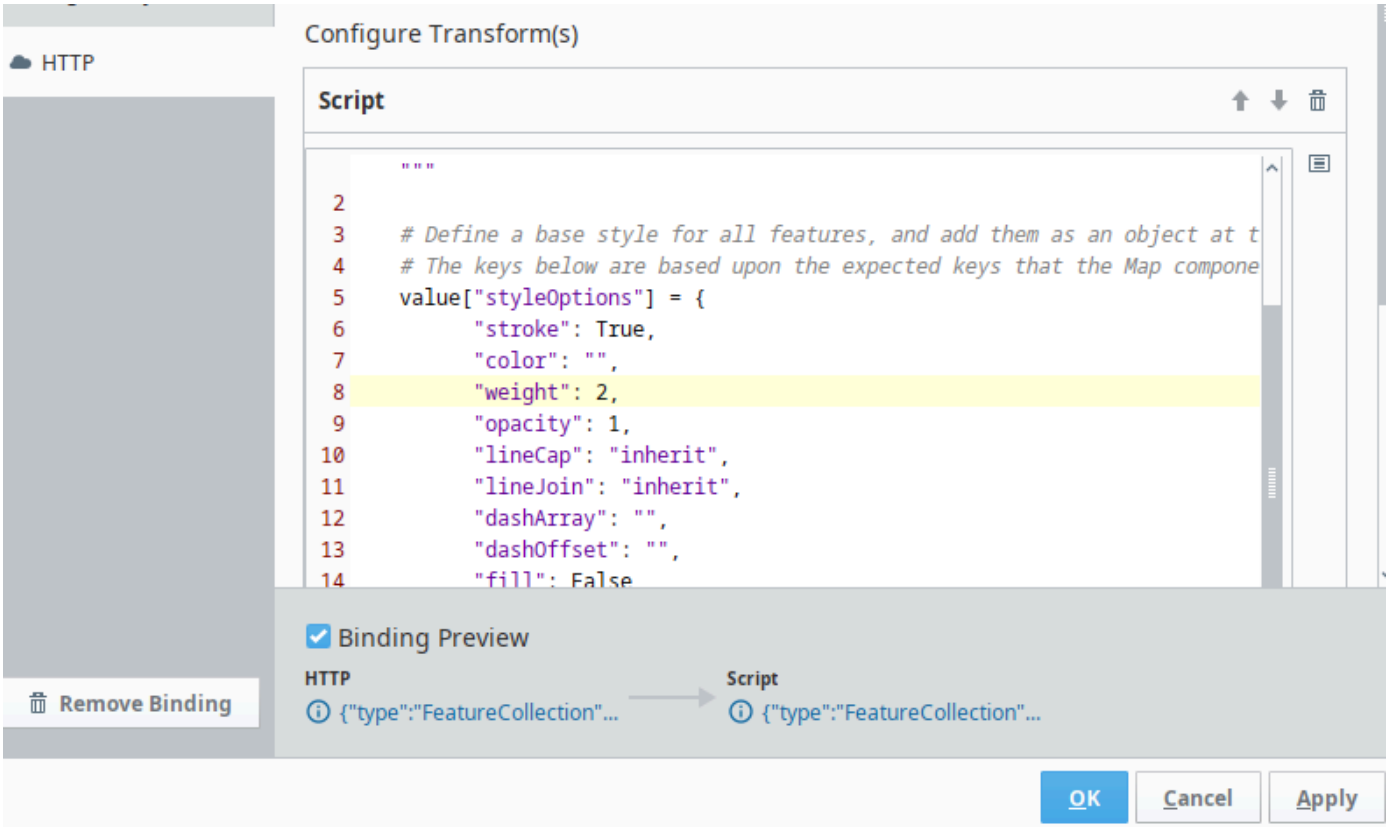
    # Note that we need to both enable fill and specify the fill color if the
feature has "fill"
    # under its raw properties key. This is because our base style above set
"fill" to False.
    if i["properties"].has_key("fill"):
        i["styleOptions"]["fillColor"] = i["properties"]["fill"]
        i["styleOptions"]["fill"] = True

    # Here we also need to change "fill-opacity" to "fillOpacity", to match
the map's default styleOptions keys.
    if i["properties"].has_key("fill-opacity"):
        i["styleOptions"]["fillOpacity"] = i["properties"]["fill-opacity"]

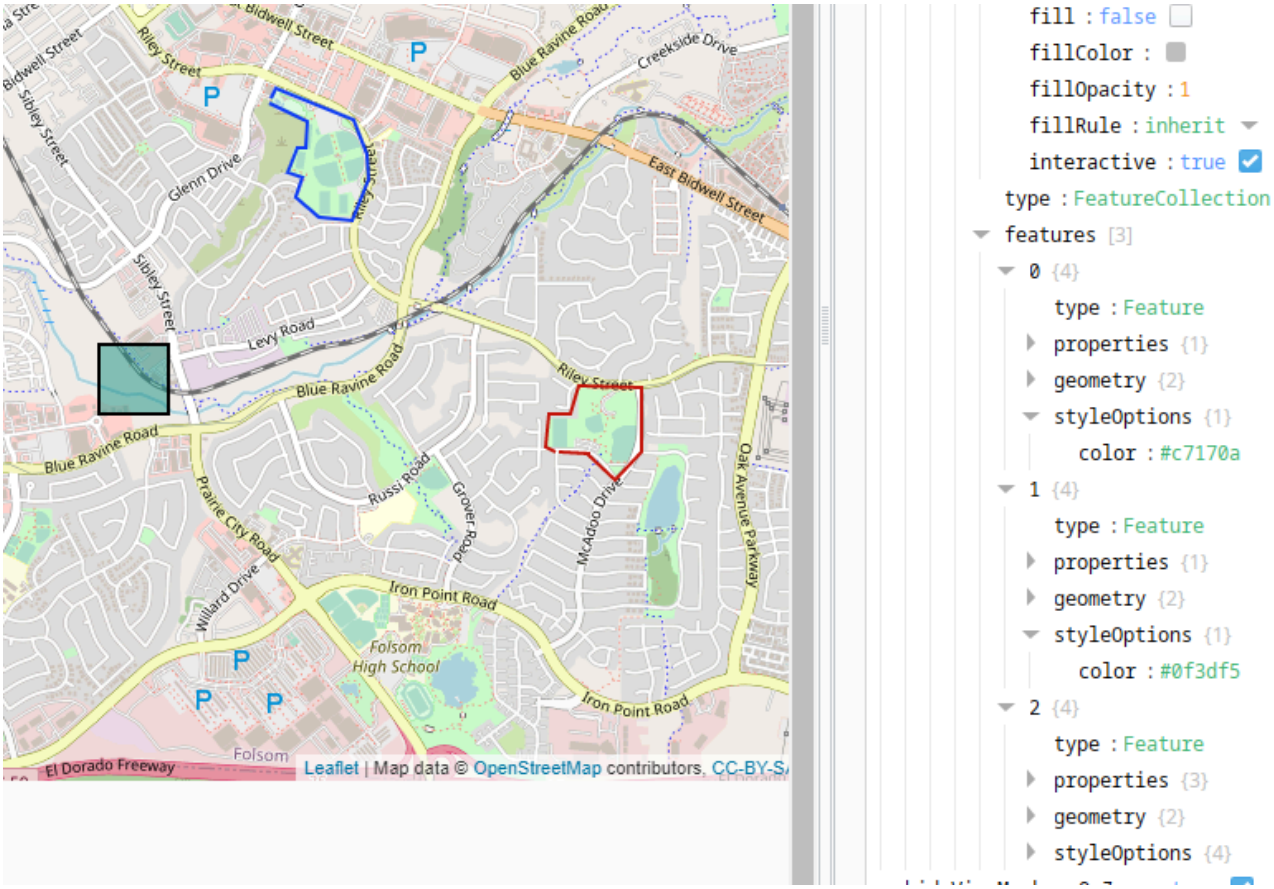

return value
```

NOTE

> Make sure the code is properly indented after pasting. No visible characters should appear on the far left of any line.



6. Click **OK** to close the binding window. Your features should now be using the updated styling data.



Adding Points

The geoJSON standard accounts for "points", which can be rendered on the map as icon based markers or simple circles. The following JSON demonstrates how to define a simple point. Note that **geometry.type** is set to "Point".

```
{
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [
      "-121.16945743560791",
      "38.6632293081453"
    ]
  }
}
```

A "marker" object can be added to points which allows for additional configuration options. For example, the following causes the point to be rendered as a circle instead of an icon.

```
{
 "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [
      "-121.16945743560791",
      "38.6632293081453"
    ]
  },
  "marker": {
    "render": "circle"
  }
}
```



Customizing Points

Points are rendered using the leaflet library, and can utilize options available in the leaflet documentation. The sections below demonstrate some available functionality. Each sub heading describes an additional object that can be added under a marker object.

Options Object

Additional settings for the marker can be applied on an **options** object under the **marker** object. A list of available options can be found in **leaflet's documentation**. The example below simply changes the opacity on the marker.

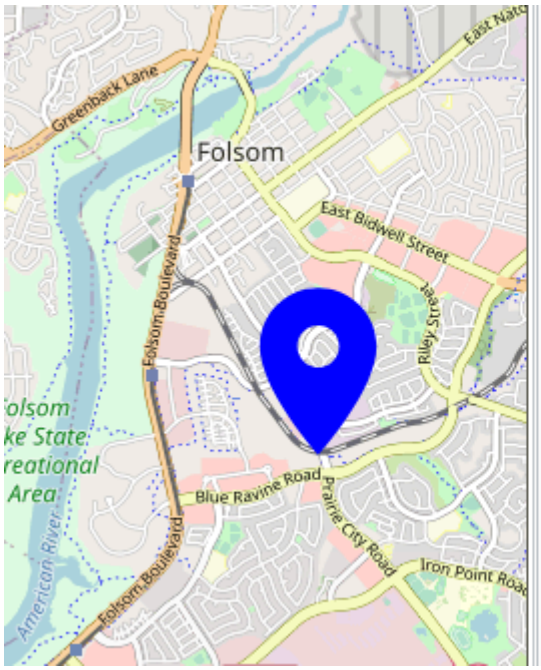
```
[
  {
    "type": "Feature",
    "properties": {},
    "geometry": {
      "type": "Point",
      "coordinates": [
        "-121.16945743560791",
        "38.6632293081453"
      ]
    },
    "marker": {
      "options": {
        "opacity": 0.5
      }
    }
  }
]
```

Icon Object

When rendered as an icon, a **icon** object can be added to configure the icon.

Setting	Description
color	A color, as represented by a string. Determines the color of the icon.
size	An array with two values: width and height. Determines the size of the icon.
path	A path to an icon. Uses the same icon library and pathing notation as the **Icon** component.

```
{
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [
      -121.16945743560791,
      38.6632293081453
    ]
  },
  "marker": {
    "icon": {
      "color": "blue",
      "size": {
        "width": 100,
        "height": 100
      },
      "path": "material/location_on"
    },
    "tooltip": {
      "content": {
        "text": "Some text"
      }
    }
  }
}
```





Tooltip Object

Tools tips can shown while hovering over a point. The text of the tooltip is driven by **marker.tooltip.content.text**.

```
[
  {
    "type": "Feature",
    "properties": {},
    "geometry": {
      "type": "Point",
      "coordinates": [
        "-121.16945743560791",
        "38.6632293081453"
      ]
    },
    "marker": {
      "tooltip": {
        "content": {
          "text": "This is my tooltip"
        }
      }
    }
  }
]
```

In addition, extra options can be added. See leaflet's documentation for a list of **tooltip options**. The example below changes the opacity on the tooltip to .5.

```
{
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [
      "-121.16945743560791",
      "38.6632293081453"
    ]
  },
  "marker": {
    "tooltip": {
      "content": {
        "text": "This is my tooltip"
      },
      "options": {
        "opacity": ".5"
      }
    }
  }
}
```

Popup Object

Simplistic popups can be added to points by adding a **marker.popup.content.text** property. Clicking such a marker will cause a popup to appear.

```
{
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [
      "-121.16945743560791",
      "38.6632293081453"
    ]
  },
  "marker": {
    "popup": {
      "content": {
        "text": "Hello from my popup"
      }
    }
  }
}
```



Like tooltips popups have additional options that can be specified in an **options** object. See leaflet's **popup documentation** for a list of possible options.

```
{
  "type": "Feature",
  "properties": {},
  "geometry": {
    "type": "Point",
    "coordinates": [
      "-121.16945743560791",
      "38.6632293081453"
    ]
  },
  "marker": {
    "popup": {
      "content": {
        "text": "Hello from my popup"
      },
      "options": {
        "minWidth": 400
      }
    }
  }
}
```

All Together

Each of these marker objects can be applied simultaneously, each adding additional functionality.




```
geometry {2}
    type : Point
  ▸ coordinates [2]
styleOptions {1}
    color : Red
marker {3}
  icon {3}
      color : blue
    ▸ size {2}
      path : material/location_on
  options {1}
      opacity : .8
  tooltip {1}
    ▸ content {1}
hideViewMarkersOnZoom : true
```

Perspective - Custom Controls Example

In this example, we'll set up three custom controls on a map component. The controls will be icons in the top right corner. When a user clicks on the home icon, the map will return to its default settings. When they click on the waves icon, the map navigates to the big island of Hawaii. When the snowflake icon is clicked, the map navigates to the Sierra Mountain range near South Lake Tahoe California.

1. Create a folder under Views called Components.

2. Next we'll create the view that will have the custom controls. In the Components folder, create a flex view called **MapLocations**. The MapLocations view will be embedded directly into the map component, and will provide some icons that users can click on to modify the map.

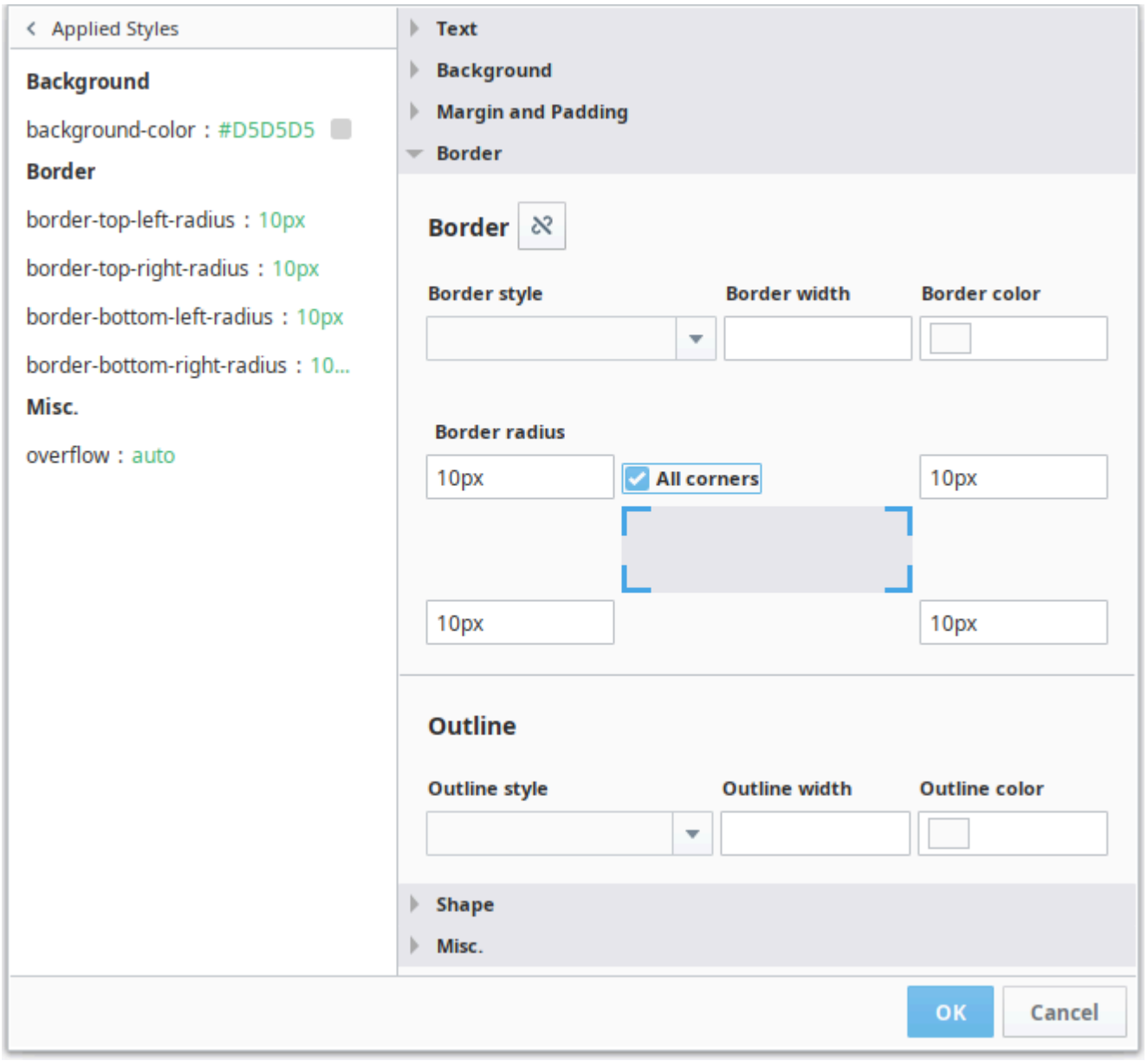
3. Set the Direction to "Row", and Justify to "Space Evenly".

4. You'll want to resize MapLocations so that it's close to the size you'd like to see it. Using the guides in the Designer, we'll set the width to about 200px and the height to about 50px.

5. To help the controls stand out from the map, let's give them a background. Select the root container, open the Style Editor, and apply the following changes:

    i. **backgroundColor**: #D5D5D5

    ii. **BoarderRadius**: Set to 10px on each corner



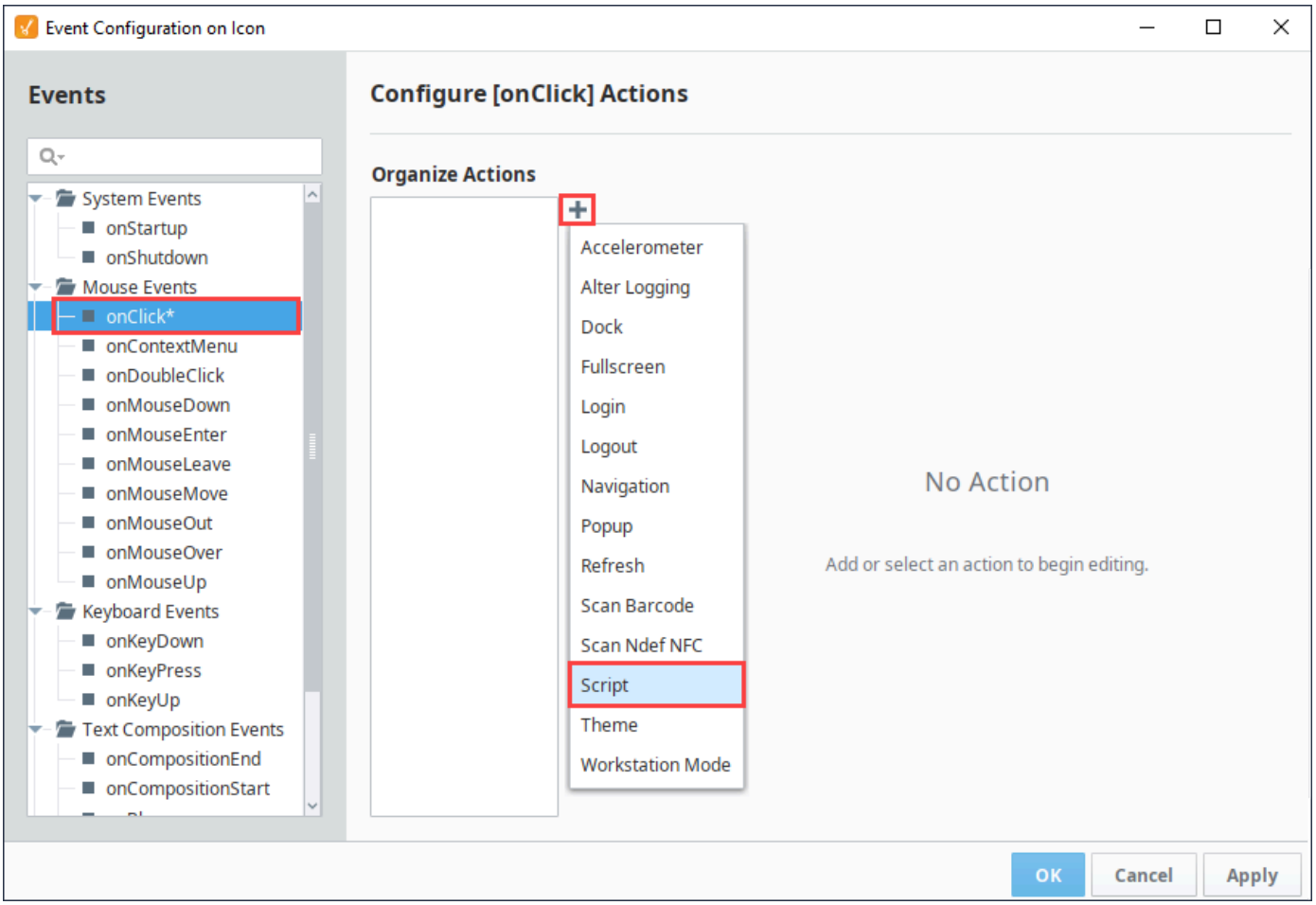
6. Drag an icon component onto the view.

    i. Set the following properties for the icon:
        - **color**: #3CACF9
        - **path**: material/home
        - **style**: Open the style editor. Under **Misc**, set **Cursor** to "Pointer"

    ii. Right-click on the icon and select **Configure Events**.

    iii. Select the Mouse Events, onClick event.

    iv. Click the **Add** ✚ icon and select the Script action:



i. In the Configure Script Action window, scroll down to the second line and enter the following:

```
system.perspective.sendMessage("NavigateMap", payload = { "lat": 38.660867,
"lng": -121.159728, "zoom": 13 }, scope = "session")
```

The script above will attempt to invoke a message handler named "NavigateMap", and pass some coordinates to it. At this point, we have not configured NavigateMap yet. We will do so in a later step.

i. Click **OK** to save the script.

ii. Next, duplicate the icon. We'll configure this icon to represent a different location. Select the first icon, copy it, and paste it. Make the following property change to it:
   - **path** : material/ waves

iii. Right-click on the second icon and select **Configure Events**.

iv. Select the Mouse Events, onClick event.

v. There should already be an action defined on this event, since our copy also has the script we wrote earlier. Select the script action.

vi. In the Configure Script Action window, scroll down to the second line and replace the "lat" and "lng" values in the payload with the following:
   - **lat**: 19.6089562306
   - **lng**: 155.385131836

The script should look like the following:

```
system.perspective.sendMessage("NavigateMap", payload = { "lat":
19.6089562306, "lng": -155.385131836, "zoom": 13 }, scope = "session")
```

vii. Click **OK** to save the script.

viii. Next, duplicate either of the icons again, and change the following property:

- **path** : material/ac_unit

ix. Right-click on the third icon and select **Configure Events**.

x. Select the Mouse Events, onClick event.

xi. Select the Script action.

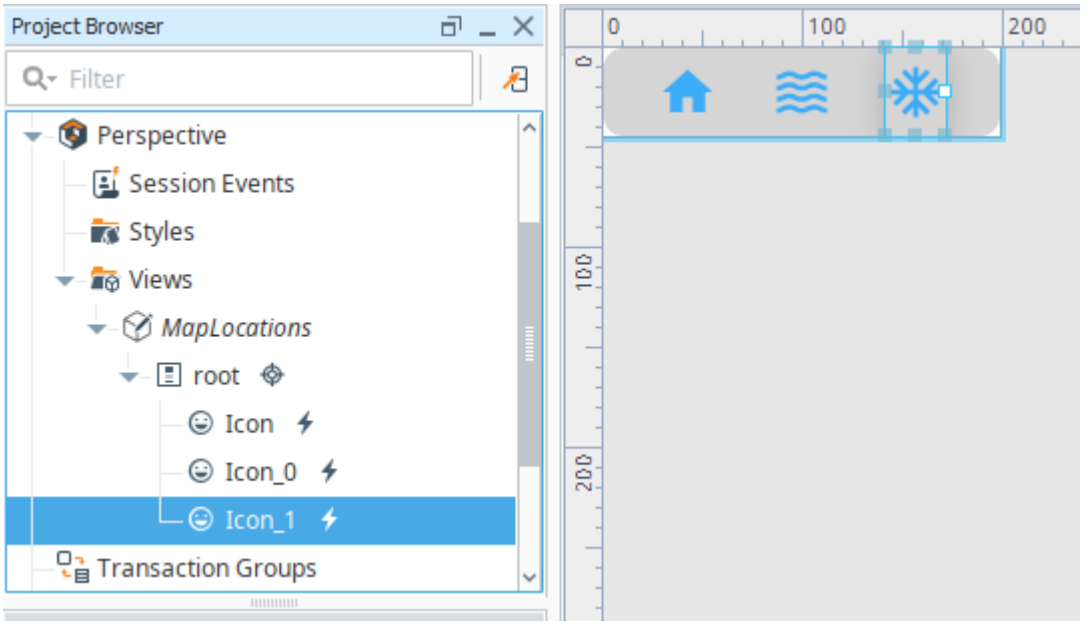
xii. In the Configure Script Action window, scroll down to the second line and replace the "lat" and "lng" values in the payload with the following:

- **lat**: 38.8665777761
- **lng**: -119.848823547

The script should look like the following:

```
system.perspective.sendMessage("NavigateMap", payload = { "lat":
38.8665777761, "lng": -119.848823547 }, scope = "session")
```

xiii. Click **OK** to save the script.



7. Next we'll set up the view with the Map component. In the Components folder, create a new Flex view and name it **Map**.

8. Drag a Map component onto the view.

9. In the Perspective Property editor, set **position.grow** on the map to 1, so the map takes up all available space.

10. Also in the Perspective Property editor, expand the **props.customControls** property.

11. Click the **Add** icon to add an array element.

12. For the props.customControls.0 properties, set the following:

    - **enabled**: true
    - **path**: Components/MapLocations
    - **position**: top-right

    You'll now see the icons from the MapLocations view appear in the upper right corner of the Map component.



13. Next, we need to configure a message handler on the Map, so when our icons the map will respond. With the Map component selected, right-click and select **Configure Scripts**.

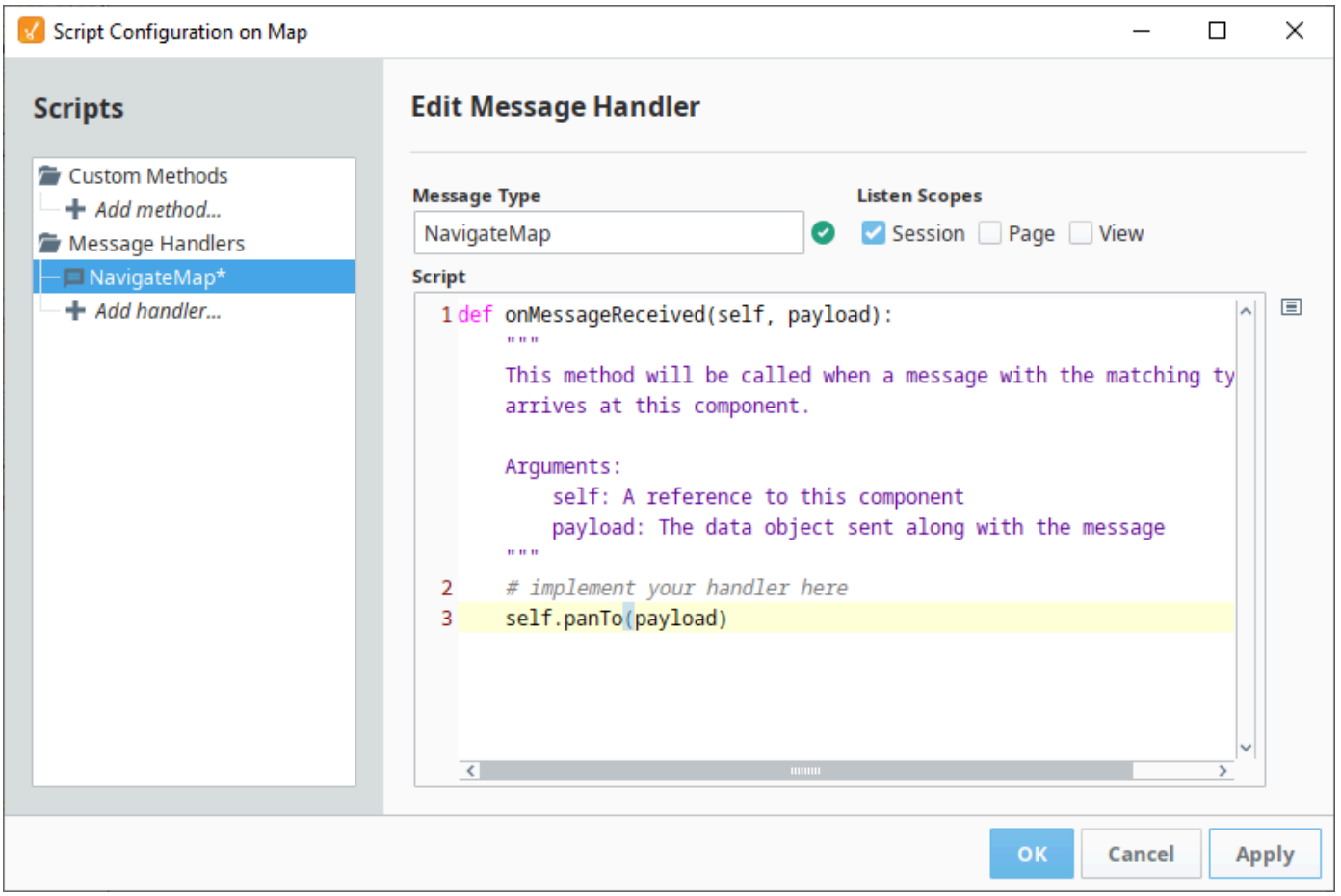
14. Double click on **Add handler** to add a new message handler.

15. Set the **Message Type** to the handler name we used in the icon scripts: NavigateMap.

16. Set **Listen Scopes** to only "Session".

17. Type the following script:

```
self.panTo(payload)
```



18. Click **OK** to commit the script.

19. Save your project.

20. Put your project into Preview mode and test it out. Click each icon and see how the location on the Map changes.

Perspective - Map Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Map** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right-clicking on the component.

onMarkerClick

Interaction event. Fired when the marker is clicked. Returns the unique name of the marker.

Object Path	Type	Description
event.name	String	The name of the marker.

onMapClick

Interaction event. Fired when the map is clicked. Returns the lat and lng of the mouse click as it translates on the map.

Object Path	Type	Description
event.lat	Numeric	The latitude of where on the map the user clicked.
event.lng	Numeric	The longitude of where on the map the user clicked.

onMapMouseMove

Interaction event. Fires as the mouse moves over the map. Returns lat and lng of the mouse as it translates on the map.

Object Path	Type	Description
event.lat	Numeric	The latitude of where the mouse moved.
event.lng	Numeric	The longitude of where the mouse moved.

onVectorClick

Interaction event. Fired when a vector is clicked. Returns the name of the vector, the type of the vector, and the coordinates of the vector.

Object Path	Type	Description
event.name	Any	The given name of the vector. Returns an empty string if a name is not defined.
event.type	String	The type of the vector clicked. Possible values are: - polygon - circle - polyline - rectangle
event.lat	Numeric	The latitude of the clicked vector.
event.lng	Numeric	The longitude of the clicked vector.

onZoomStart

Map state event. Fired when the map zoom is about to change (e.g. before zoom animation). Returns the zoom level.

Object Path	Type	Description
event.zoom	Numeric	What the zoom level on the map was before zooming.

onZoom

Map state event. Fired repeatedly during any change in zoom level, including zoom and fly animations. Returns the zoom level.

Object Path	Type	Description
event.zoom	Numeric	What the zoom level was changed to.

onZoomEnd

Map state event. Fired when the map has changed, after any animations. Returns the zoom level.

Object Path	Type	Description
event.zoom	Numeric	What the zoom level on the map was after zooming.

onMoveStart

Map state event. Fired when the view of the map starts changing (e.g., user starts dragging the map). Returns the map center as lat and lng.

Object Path	Type	Description
event.lat	Numeric	The latitude of the center of the map before moving.
event.lng	Numeric	The longitude value for the center of the map before moving.

onMove

Map state event. Fires repeatedly during any movement on the map, including pan and fly animations. Returns the map center as lat and lng.

Object Path	Type	Description
event.lat	Numeric	The latitude value for the center of the map during moving.
event.lng	Numeric	The longitude value for the center of the map during moving.

onMoveEnd

Map state event. Fired when the center of the map stops changing (e.g. user stopped dragging the map). Returns the new map center as lat and lng.

Object Path	Type	Description
event.lat	Numeric	The latitude value for the center of the map after moving.
event.lng	Numeric	The longitude value for the center of the map after moving.

onResize

Map state event. Fires when the map size has changed. Returns the map size as oldSize and newSize.

Object Path	Type	Description
event.oldSize	PyJsonObjectAdapter	Returns the starting size of the map before the resize. Values are returned in an object that's functionally similar to a Python dictionary. The object contains two values: height and width.
event.newSize	PyJsonObjectAdapter	Returns the starting size of the map after the resize. Values are returned in an object that's functionally similar to a Python dictionary. The object contains two values: height and width.

Component Functions

.getCenter()

- Description
  - Returns the geographical center of the map in latitude and longitude.
- Parameters
  - Nothing

- Return
    - LatLng - Returns the geographical center of the map view as `{"lat": number, "lng": number}`.

.getZoom()

- Description
    - Returns the current zoom level of the map view as a number.
- Parameters
    - Nothing
- Return
    - Returns the current zoom level of the map view as a number.

.getBounds()

- Description
    - Returns the geographical bounds of the map as a dictionary.
- Parameters
    - Nothing
- Return
    - Dictionary: A dictionary containing the following keys:
      `north: number`
      `northEast: LatLng`
      `east: number`
      `southEast: LatLng`
      `south: number`
      `southWest: LatLng`
      `west: number`
      `northWest: LatLng`

.getBoundsAsBBoxString()

- Description

    - Returns a string with bounding box coordinates in a 'South West longitude, South West latitude, North East longitude, North East latitude' format.

- Parameters
    - Nothing
- Return
    - Returns the bounding box of the map as a string.

.zoomIn([delta, options])

- Description
    - Increases the zoom of the map by delta.
- Parameters
    - Numeric **delta** - The numerical value to increase the zoom by. If omitted, uses the value of props.zoom.delta. [optional]
    - Dictionary **options** - A dictionary of parameters to use during the zoom, typically containing a single key, animate. See **Perspective - Map ZoomOptions**. [optional]
- Return
    - Nothing

.zoomOut([delta, options])

- Description
    - Decreases the zoom of the map by delta.
- Parameters
    - Numeric **delta** - The numerical value to increase the zoom by. If omitted, uses the value of props.zoom.delta. [optional]
    - Dictionary **options** - A dictionary of parameters to use during the zoom, typically containing a single key, animate. See **Perspective - Map ZoomOptions**. [optional]
- Return
    - Nothing

.setZoomAround(point, zoom, [options])

- Description
  - Zooms the map while keeping a specified geographical point on the map stationary (e.g. used internally for scroll zoom and double-click zoom)
- Parameters
  - Dictionary **point** - The geographic point that the map will zoom around. See **Perspective - Map LatLng**. [required]
  - Numeric **zoom** - The numerical value to increase the zoom by. If omitted, uses the value of props.zoom.delta. [optional]
  - Dictionary **options** - A dictionary of parameters to use during the zoom, typically containing a single key, animate. See **Perspective - Map ZoomOptions**. [optional]
- Return
  - Nothing

.fitBounds(latLngBounds, [options])

- Description
  - Sets a map view that contains the given geographical bounds with the maximum zoom level possible
- Parameters
  - Dictionary **latLngBounds** - A dictionary consisting of two LatLng objects. The LatLng objects combined represent the geographical bounds the map view should be set to. [required]
  - Dictionary **options** - A dictionary of parameters used to manipulate the FitBound settings. See **Perspective - Map FitBounds**. [optional]
- Return
  - Nothing

.fitWorld([options])

- Description
  - Sets a map view that mostly contains the whole world with the maximum zoom level possible.
- Parameters

    - Dictionary **options** - A dictionary of parameters used to manipulate the FitBound settings. See **Perspective - Map FitBounds**. [optional]

- Return
    - Nothing

.panTo(latLng, [options])

- Description
    - Pans the map to a given center.
- Parameters
    - Dictionary **latLng** - The geographic point to pan to. [required]
    - Dictionary **options** - A dictionary of parameters used to modify the panning behavior. See **Perspective - Map PanOptions**. [optional]
- Return
    - Nothing

.panBy(point, [options])

- Description
    - Pans the map by a given number of pixels (animated).
- Parameters
    - Dictionary **point** - The geographic point to pan to. The dictionary should contain an 'x' and 'y' key, both with numeric values. [required]
    - Dictionary **options** - A dictionary of parameters used to modify the panning behavior. See **Perspective - Map PanOptions**. [optional]
- Return
    - Nothing

.flyTo(latLng, [zoom, options])

- Description

    - Sets the view of the map (geographical center and zoom) performing a smooth pan-zoom animation.

- Parameters

    - Dictionary **latLng** - A Python dictionary representing the coordinates to fly to. [required]
    - Numeric **zoom** - Sets the zoom level to transition to during the flight. If omitted, uses the value of props.zoom.delta. [optional]
    - Dictionary **options** - A dictionary of panning options to use. See **Perspective - Map PanOptions**. [optional]

- Return

    - Nothing

.flyToBounds(latLngBounds, [options])

- Description

    - Sets the view of the map with a smooth animation like flyTo, but takes a bounds parameter like FitBounds.

- Parameters

    - Dictionary **latLngBounds** - A dictionary consisting of two LatLng objects. The LatLng objects combined represent the geographical bounds the map view should be set to. [required]
    - Dictionary **options** - A dictionary of panning options to use. See **Perspective - Map PanOptions**. [optional]

- Return

    - Nothing

.panInsideBounds(latLngBounds, [options])

- Description

    - Pans the map to the closest view that would lie inside the given bounds (if it's not already), controlling the animation using the options specific, if any.

- Parameters

    - Dictionary **latLngBounds** - A dictionary consisting of two LatLng objects. The LatLng objects combined represent the geographical bounds the map view should be set to. [required]

    - Dictionary **options** - A dictionary of panning options to use. See **Perspective - Map PanOptions**. [optional]

- Return
    - Nothing

.panInside(latLng, [options])

- Description
    - Pans the map the minimum amount to make the latLng visible.
- Parameters
    - Dictionary **latLng** - A Python dictionary representing the coordinates to pan to. [required]
    - Dictionary **options** - A dictionary of panning options to use. See **Perspective - Map PanOptions**. [optional]
- Return
    - Nothing

.getSize()

- Description
    - Returns height and width of the Map component.
- Parameters
    - Nothing
- Return
    - JSON Object - Returns a Python dictionary. Contains two items: height and width.

Scripting Example

```
def doMapStuff(self):
    map = self.getSibling("Map")
    coordinateBounds = {'corner1': {'lat': 39.086798, 'lng': -120.069014},
'corner2': { 'lat': 38.815319, 'lng': -119.787519 }}
    latLngTahoe = {'lat': 39.086798, 'lng': -120.069014 }
    latLngInductive = {'lat': 38.652511, 'lng': -121.189438 }
    zoomPanOptions = { 'animate': True, 'duration': 3, 'easeLinearity': 0.25,
'noMoveStart': False }
    fitBoundsOptions = { 'padding': { 'x': 100, 'y': 100 }, 'animate': True,
'duration': 3 }
    panPixels = { 'x': 200, 'y': 200 }

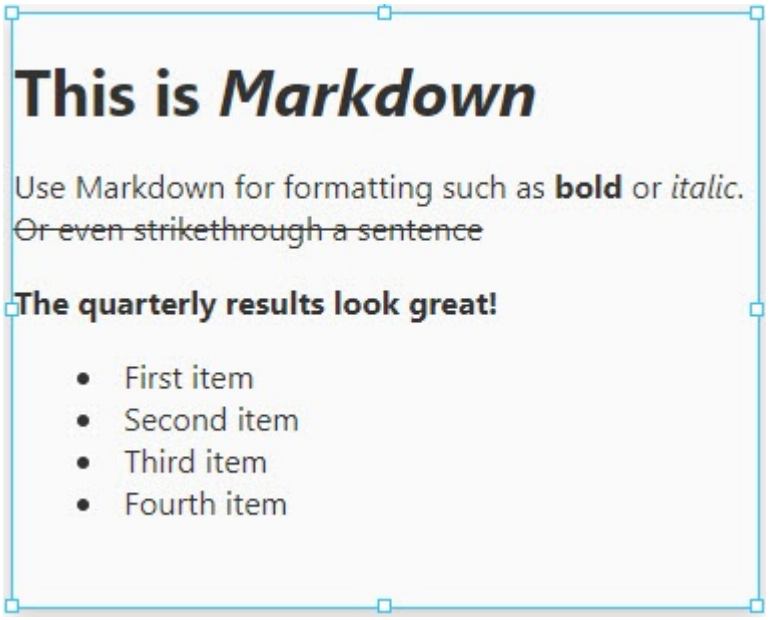

    print map.getCenter()
    # Returns the geographical bounds visible in the current map view as
latLngBounds

    map.flyTo(latLng = latLngTahoe, options = zoomPanOptions)
    # Sets the view of the map with a smooth animation like flyTo, but takes a
bounds parameter like FitBounds
```

Extension Functions

This component does not have extension functions associated with it.

Perspective - Markdown



Component Palette Icon:

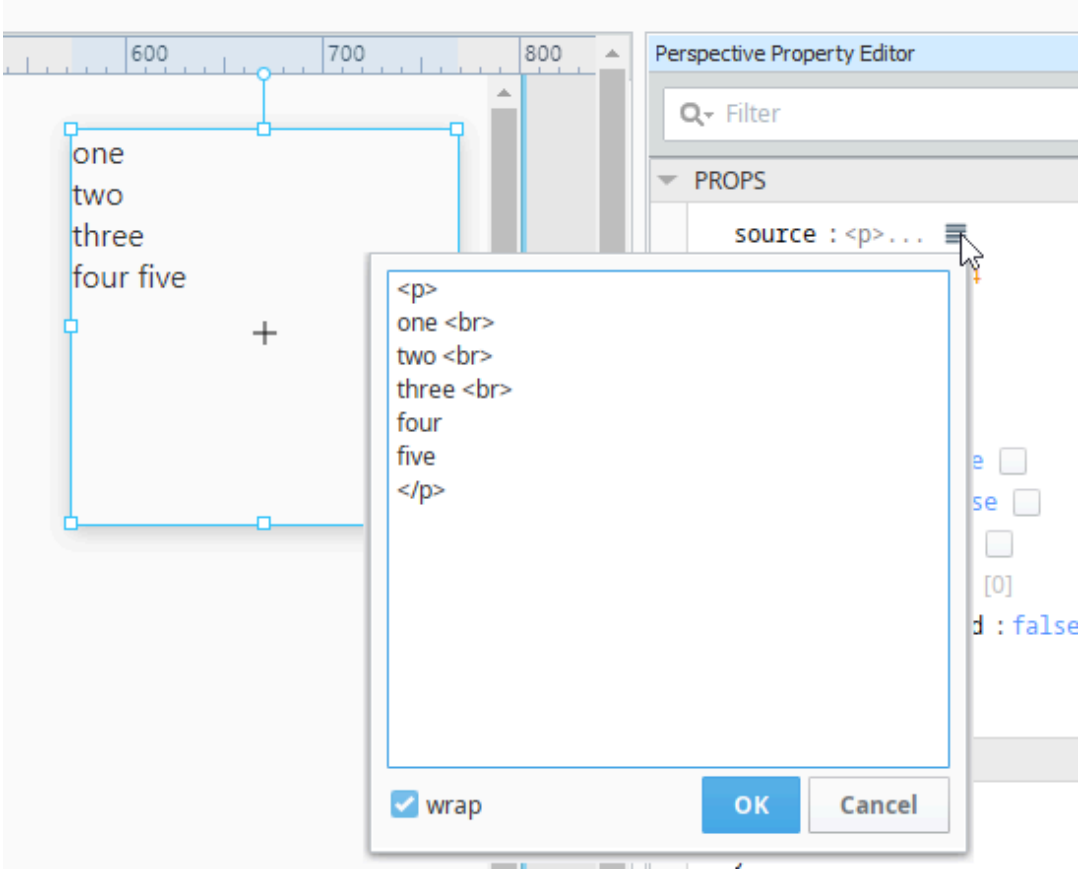


Description

The Markdown component allows users to format any type of text so it is publishable as plain text without looking like it's been marked up with tags or formatting instructions. The Markdown component provides is a lightweight formatting language which is easy to write and easy to read. Markdown's formatting syntax only addresses issues that can be conveyed in plain text. For any marked up content that is not covered by Markdown's syntax, you can use HTML. You can even change the color of text in the component with HTML tags using the `source` prop in the Property Editor as shown in Example 2.

To learn more about how to use Markdown component for publishing plain text, refer to the following articles: **Markdown Basics** and **Markdown Node Types**.

If you would like to manually add line breaks, the most direct approach would be to disable the escapeHTML property, and add your own paragraph and line break elements. Once disabled, you can add line breaks as seen below:

Value of `source`	Result
 one two three four five 	Because there isn't a line break between "four" and "five", they're on the same line. 

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
source	Text annotated with markdown syntax to display.	value: string
sectionSpacing	Number of pixels of vertical space between each section or header.	value: numeric
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
markdown	Lightweight formatting language that is easy to write and easy to read. • **sourcePos**: If true, will keep track and log source positioning for debugging purposes. Default is false. Value is boolean. • **escapeHtml**: Setting to false will cause HTML to be rendered. Be aware that setting this to false might cause security issues if the input is user-generated. Use at your own risk. Default is true. Value is boolean. • **skipHtml**: Setting to true will skip inlined and blocks of HTML. Default is false. Value is boolean. • **disallowedTypes**: Defines which types of types of nodes should be disallowed (not rendered). Value is array. • **unwrapDisallowed**: Setting to true will try to extract/unwrap the children of disallowed nodes. For instance, if disallowing Strong, the default behavior is to simply skip the text within the strong altogether, while the behavior some might want is to simply have the text returned without the strong wrapping it. Default is false. Value is boolean. • **renderers**: Renders in a browser as plain text.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Markdown **basic syntax** link.
From https://www.markdownguide.org.
Text formatting examples:

Heading 1

Heading 2

Heading 3

Bold text

Italic text

Bold and Italic text

Line of plain text

Many other formatting options are listed online.

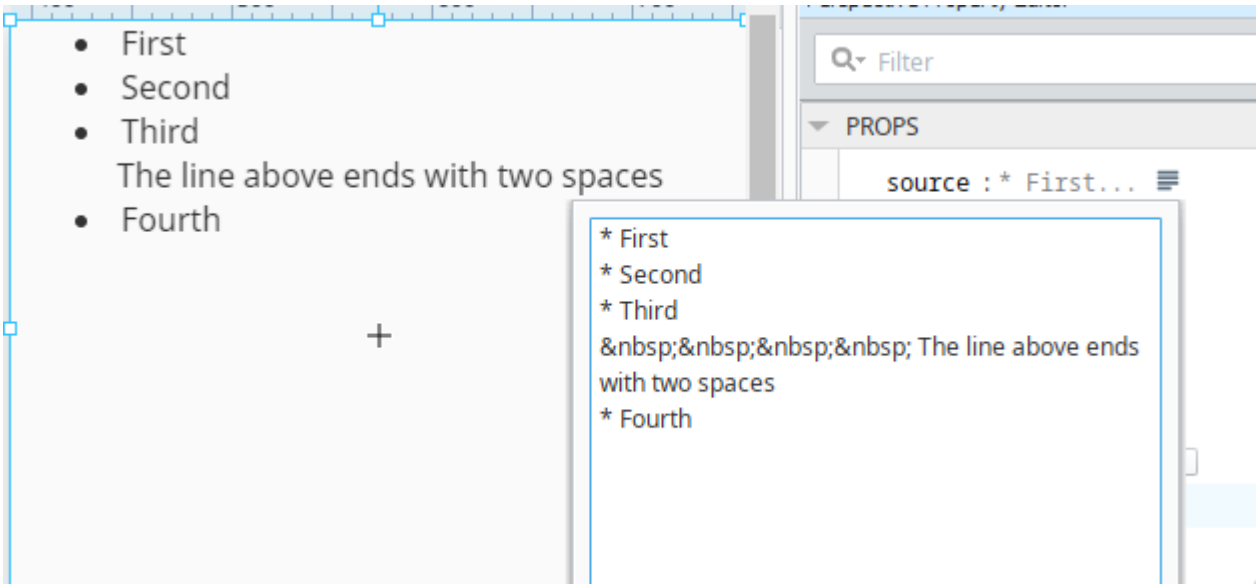
Property	Value

props.source	MARKDOWN source:
	`Markdown basic syntax link.`
	
	`From https://www.markdownguide.org.`
	
	`Text formatting examples:`
	`# Heading 1`
	`## Heading 2`
	`### Heading 3`
	`**Bold text**`
	
	`*Italic text*`
	
	`***Bold and Italic text***`
	
	`Line of plain text`
	
	`Many other formatting options are listed online.`

Example 2

You can add additional lines within a list by ending the previous line with two spaces. In the example below, the text on the third line is actually "Third ". You can then force spaces via HTML using ` `.

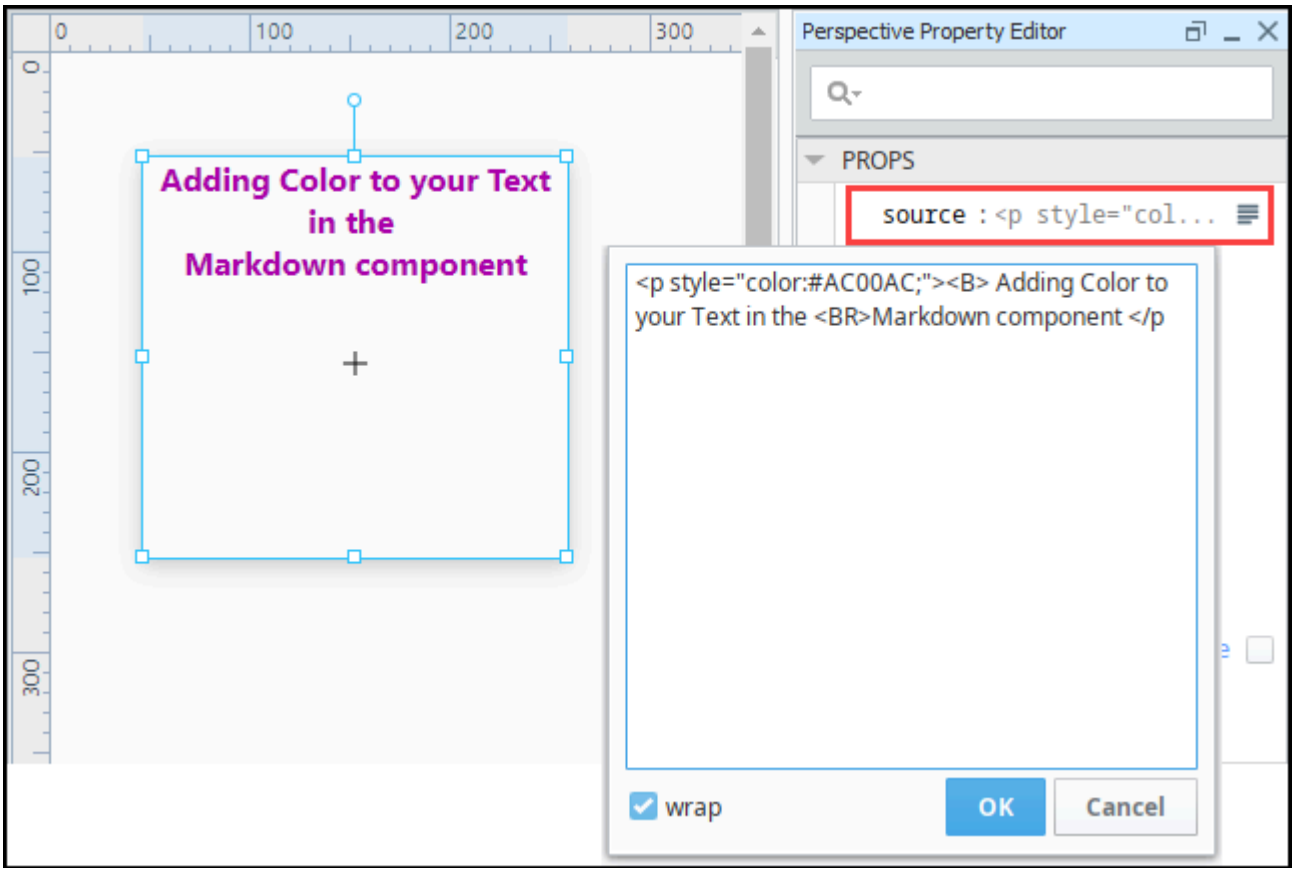
```
* First
* Second
* Third
    The line above ends with two spaces
* Fourth
```



Example 3

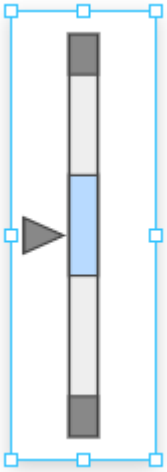
The following code was used in the example below and pasted into the `source` property of the Markdown component.

```
<p style="color:#AC00AC;"><B> Adding Color to your Text in the <BR>Markdown component </p
```

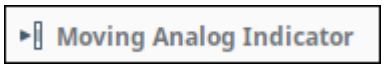


Adding Color to your Text in the Markdown component

Perspective - Moving Analog Indicator



Component Palette Icon:



Description

The Moving Analog Indicator displays an analog value in context with other information about that value so that you can visually and quickly see if the value is in the normal range or not. The current value is shown as an arrow pointing at a bar with segments showing the desired operating range, low and high alarm ranges, and interlock ranges.

The Moving Analog Indicator component allows for extremely fast information delivery. At a glance, it is obvious to an operator whether or not the value is where it should be, or if it needs attention. If the value is in one of its alarm ranges, then that range can be set to change color to get attention.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
processValue	Current value of the process.	value: numeric
setpointValue	Current value of the setpoint.	value: numeric
minValue	The minimum value shown on the indicator. Default is 0.	value: numeric
maxValue	The maximum value shown on the indicator. Default is 100.	value: numeric
desiredHigh	The upper limit of the desired range. Default is 65.	value: numeric
desiredLow	The lower limit of the desired range. Default is 40.	value: numeric
highAlarm	Value above indicating a high alarm. Default is 90.	value: numeric
highHighAlarm	Value above indicating a high-high alarm. Default is null.	value: numeric
highInterlock	Value above when an interlock will be activated. Default is null.	value: numeric
lowAlarm	Value below indicating a low alarm. Default is 10.	value: numeric
lowLowAlarm	Value below indicating a low-low alarm. Default is null.	integer
lowInterlock	Value below when an interlock will be activated. Default is null.	integer

Name	Description	Property Type
desiredRangeColor	Color for the area in the desired range. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector.**	color
defaultRangeColor	Color for the area not defined as a range. See **Color Selector.**	color
inactiveAlarmColor	Color for the inactive alarm range. See **Color Selector.**	color
level2AlarmColor	Color for the active level 2 alarm (high or low). See **Color Selector.**	color
level1AlarmColor	Color for the active level 1 alarm (high-high or low-low). See **Color Selector.**	color
interlockColor	Color for the interlock range. See **Color Selector.**	color
indicatorColor	Color for the process indicator value. See **Color Selector.**	color
setpointColor	Color for setpoint value marker. See **Color Selector.**	color
label	Numeric value displayed as text next to the indicator. • **visible**: Whether to display the label. Default is false. Value is boolean. • **format**: Format of numeric value in label, including commas, decimal places, etc. Value is a string dropdown. Options as follows: ◦ #,##0 ◦ #,##0.0 ◦ #,##0.00 ◦ 0 ◦ 0.0 ◦ 0.00 ◦ #,##0% • **style** Sets a style for the label property. Full menu of **style options** is available for text, background, margin	object

Name	Description	Property Type
	and padding, border, shape and miscellaneous. You can also specify a **style class**.	
selectOutline	Border settings for the outline surrounding each range area. Options as follows: • **color**: Sets the color for the outline surrounding the range borders. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. • **width**: Sets the width of the outline surrounding the range borders in pixels. Value is numeric.	object
reverseIndicator	Displays the process value indicator on the opposite side of the scale. Default is false.	value: boolean
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

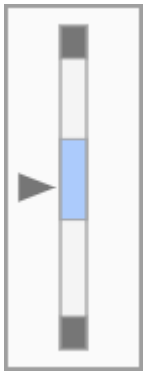
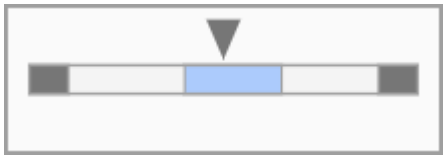
Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

The alignment of the component is based on the height and width of the component.

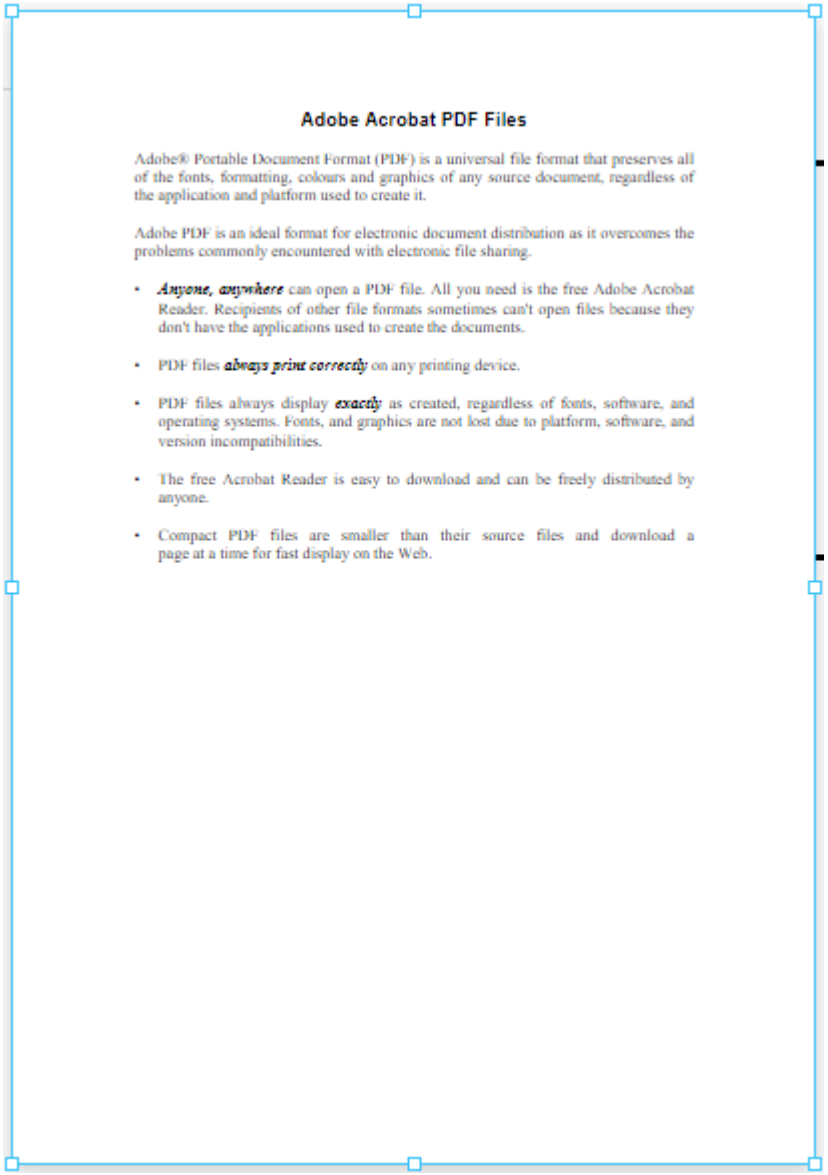
Height > Width	Width > Height
	

Example 2

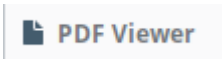


Property	Value
props.processValue	96
props.highHighAlarm	96
props.reverseIndicator	true
props.label.visible	true

Perspective - PDF Viewer



Component Palette Icon:



Description

The PDF Viewer component displays a PDF that's hosted on a web server by providing a URL to the source property. A simple approach is to create either a File Resource or Mounted Folder within **Web Dev**, and set the source on the component to the resource's endpoint.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
showPageNumber	If true, the current page number and page count will be shown at the bottom of the component.	value: boolean
source	Path to the .pdf file to be displayed. Expects a URL to a PDF hosted on a web server.	value: string
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
page	The current page being displayed.	value: numeric
pageCount	The number of pages the pdf contains. Read only.	value: numeric

Scripting

See the **Perspective - PDF Viewer Scripting** page for the full list of scripting functions available for this component.

Perspective - PDF Viewer Scripting

This page details the various scripting, component, and extension functions available for Perspective's **PDF Viewer** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Component Functions

.reload()

- Description
  - This function will reload the PDF in the PDF Viewer component.
- Parameters
  - String **name** - The name of the PDF.
- Return
  - Nothing

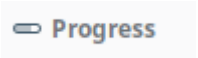
Extension Functions

This component does not have extension functions associated with it.

Perspective - Progress



Component Palette Icon:



Description

The Progress bar visually indicates the progress of a task. It is used to display any value that has an upper and lower bound. Custom settings are available for the track and the bar.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
min	The minimum value of the progress indicator. If the value reaches the max, the progress indicator will be completely filled. Must be less than the max. Default is 0.	value: numeric
max	The maximum value of the progress indicator. If the value reaches the max, the progress indicator will be completely filled. Must be greater than 0.0. Default is 100.	value: numeric
value	The current value representing the current progress. Must be greater than 0.0 and less than the value set in max. Default is 50.	value: numeric
mode	Determines if the component should show a determinate state, or an indeterminate loading state. When set to determinate, shows the progress of the value relative to the min and max properties.	value: string
bar	Settings for the bar. Full property descriptions included **below**.	object
track	Settings for the track. Full property descriptions included **below**.	object
valueDisplay	Value display configuration. Renders and styles a value overlay above the progress bar. • **enabled**: If true, will show the value display. Value is boolean. • **format**: Format to apply to value, which is then used in the value display. Value is string. • **justify**: Horizontal alignment of the displayed value. Value is string. • **style**: Sets a style for the track. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Name	Description	Property Type
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

bar and track Child Properties

Name	Description	Property Type
color	A convenience property for setting the base color. This can also be accomplished by using the `prop.bar.style` or `prop.track.style` to set the background color.	object
style	Sets a style. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
determinate	Determinate configuration. • **color**: A convenience property for setting the color when mode is determinate. This can also be accomplished by using the `prop.bar.determinate.style` or `prop.track.determinate.style` to set the background color. • **style**: Sets a style for the bar when mode is determinate. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
indeterminate	Indeterminate configuration. • **color**: A convenience property for setting the color when mode is indeterminate. This can also be accomplished by using the `prop.bar.indeterminate.style` or `prop.track.indeterminate.style` to set the background color. • **style**: Sets a style for the bar when mode is indeterminate. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1



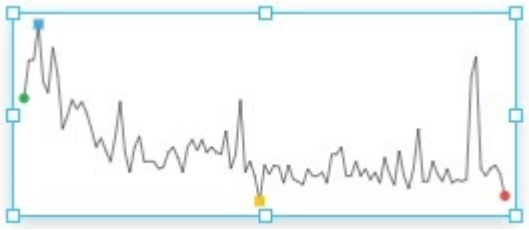
Property	Value
max	100
value	80

Example 2

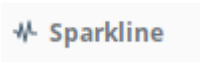


Property	Value
max	100
value	56
mode	determinate
bar.color	#FFAC47
track.color	#FFE8CC
track.borderStyle	solid
track.borderWidth	2
track.borderBottomLeftRadius	15
track.borderBottomRightRadius	15
track.borderTopRightRadius	15
track.borderTopLeftRadius	15
track.borderColor	#A45324
valueDisplay.enabled	true
valueDisplay.format	percent
valueDisplay.justify	center
valueDisplay.fontFamily	Merriweather
valueDisplay.fontWeight	bold

Perspective - Sparkline



Component Palette Icon:



Description

The Sparkline chart is a minimalistic chart component that displays a line-chart history for a single datapoint. Sparklines were invented by Edward Tufte as a way to show a great deal of contextual information in a very small amount of space. Sparklines are typically used to display the recent history (up to current time) of a datapoint so that the viewer can quickly discern the recent trend of a datapoint: is it rising? falling? oscillating? etc..

To use a sparkline, bind its Data property either to a Historian realtime query, or to a database query. There should be two columns in this dataset: the first one a date column, the second a number. Each row will become a datapoint on the chart, and the dataset must be sorted by time in ascending order.

Instead of using axes to convey scale, the Sparkline can display a band of color across the back of the chart which indicates the desired operating range of the datapoint. In this way, it is instantly obvious when a value is in its expected range, above that range, or below. The sparkline automatically configures its internal axes based on the data given to it. To display a desired range, fill in the **props.desired.high** and **props.desired.low** properties.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
points	Data points to plot. Accepts the following: May be a dataset, or an array of values or of objects containing X or Y coordinates. Also may be a string formatted with X and Y values separated by a comma. • An array of numbers containing X or Y coordinates. • A string of space delimited points where x and y are separated by a comma. For example: 0,20 1,35 2,15. • An array of objects, where each object contains an x and a y property, and where each property's value is a number. • A dataset of a single column of number type. • A dataset of two columns, the first representing the x value and the second column representing the y value. The first column can be either of type number or type date. Dates and Timestamps are converted to unix timestamps which is used as the x value.	array or dataset
color	Color of the line. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	color
width	Thickness of the line, in pixels.	value: numeric
opacity	The opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	value: numeric
dashArray	The pattern of dashes and gaps used to paint the stroke. It's a list of comma separated lengths (in pixels) and percentages (percentage of the total stroke length) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5,3,2" is equivalent to "5,3,2,5,3,2".	array

Name	Description	Property Type
range	Settings for the upper and lower edge of the chart. • **high**: A fixed value for the upper edge of the chart as a number. • **low**: A fixed value for the lower edge of the chart as a number.	object
desired	The desired operating range. Settings for the desired properties operating range. • **high**: The high value of the desired operating range. Value is numeric. • **low**: The low value of the desired operating range. Value is numeric. • **stroke**: Settings for the stroke. Options as follows: ◦ **color**: Color of the line. See **Color Selector**. ◦ **width**: Width of the line in pixels. Value is numeric. ◦ **opacity**: Opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. Value is numeric. ◦ **dashArray**: The pattern of dashes and gaps used to paint the stroke. It's a list of space separated lengths (in pixels) and percentages (percentage of the stroke length for the desired operating range) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5 3 2" is equivalent to "5 3 2 5 3 2". • **fill**: Settings for the fill for the desired range. Options as follows: ◦ **color**: Fill color of the desired range. See **Color Selector**. ◦ **opacity**: Opacity of the fill ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	object
marker	Settings for the markers on the chart. Each of the following markers have the child properties listed in the **marker Type Child Properties table** below. • **first**: Settings for the first marker on the chart.	object

Name	Description	Property Type
	• **last**: Settings for the last marker on the chart. • **high**: Settings for the high marker on the chart. • **low**: Settings for the low marker on the chart.	
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

marker Type Child Properties

The following properties apply to the first, last, high, and low markers on the chart.

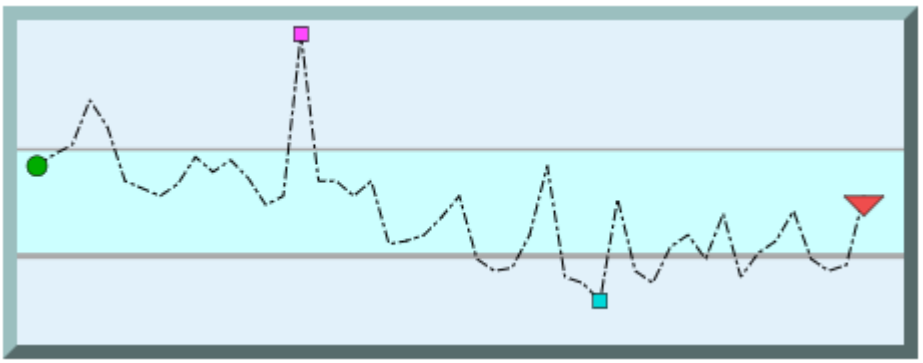
Name	Description	Property Type
shape	Shape of the marker used to indicate the first point. Options are circle, triangle, or square. Default is circle.	value: string dropdown
size	Size of the marker, in pixels.	value: numeric
stroke	Settings for the stroke for the first marker. Options as follows: • **color**: Color of the line. See **Color Selector**. • **width**: Width of the line in pixels. Value is numeric. • **opacity**: Opacity of the line ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque. Value is numeric. • **dashArray**: The pattern of dashes and gaps used to paint the stroke. It's a list of space separated lengths (in pixels) and percentages (percentage of the stroke length for the desired operating range) that specify the lengths of alternating dashes and gaps. If an odd number of values is provided, then the list of values is repeated to yield an even number of values. Thus, "5 3 2" is equivalent to "5 3 2 5 3 2".	object
fill	Settings for the fill for the first marker. Options as follows: • **color**: Fill color of the desired range. See **Color Selector**. • **opacity**: Opacity of the fill ranging from 0 to 1. 0 is fully transparent, 1 is fully opaque.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

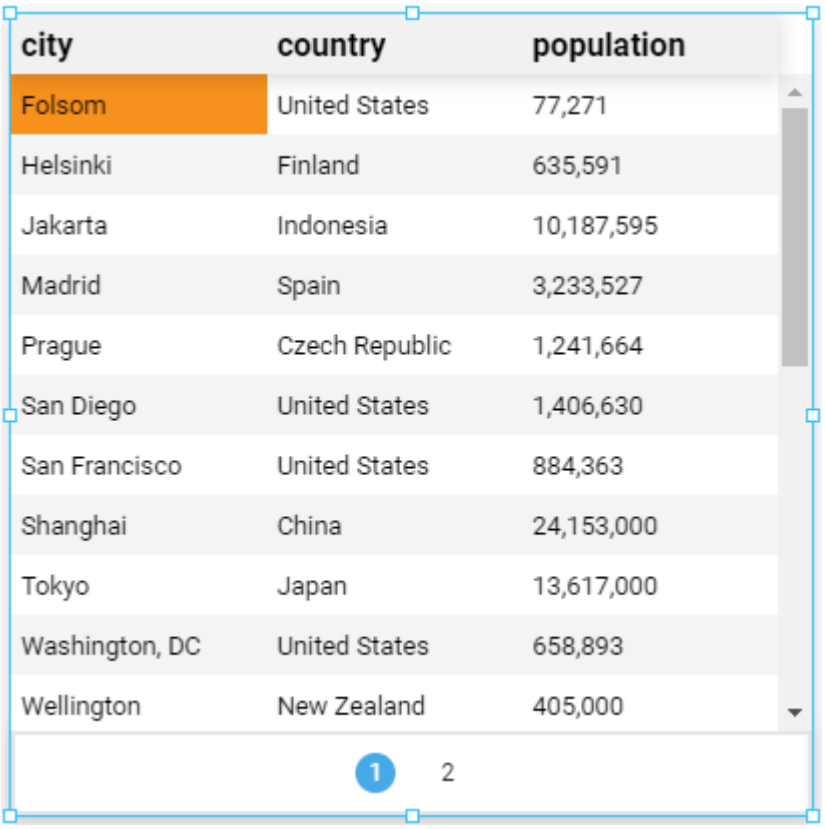
This example demonstrates what changes to the various marker properties can do. In addition, the desired range has been enabled.



Property	Value
props.width	1
props.dashArray	6 2 2 2
props.desired.high	60
props.desired.low	25
props.desired.stroke.color	#AAAAAA
props.desired.stroke.width	3
props.desired.stroke.opacity	3
props.desired.fill.color	#CCFFF
props.desired.fill.opacity	4
props.marker.first.shape	circle
props.marker.first.size	10
props.marker.first.stroke.color	#000000
props.marker.first.fill.color	#00AC00
props.marker.last.shape	triangle
props.marker.last.size	20
props.marker.last.stroke.color	#000000
props.marker.last.fill.color	#EF4D4D
props.marker.low.shape	square
props.marker.low.size	7
props.marker.low.stroke.color	#000000

Property	Value
props.marker.low.fill.color	#0D9D9
props.marker.high.shape	square
props.marker.high.size	7
props.marker.high.stroke.color	#000000
props.marker.high.fill.color	#FF47FF
props.style.backgroundColor	#E2F1Fa
props.style.borderStyle	outset
props.style.borderWidth	7px
props.style.borderColor	#9BBFBF

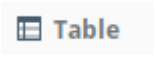
Perspective - Table



city	country	population
Folsom	United States	77,271
Helsinki	Finland	635,591
Jakarta	Indonesia	10,187,595
Madrid	Spain	3,233,527
Prague	Czech Republic	1,241,664
San Diego	United States	1,406,630
San Francisco	United States	884,363
Shanghai	China	24,153,000
Tokyo	Japan	13,617,000
Washington, DC	United States	658,893
Wellington	New Zealand	405,000

1 2

Component Palette Icon:

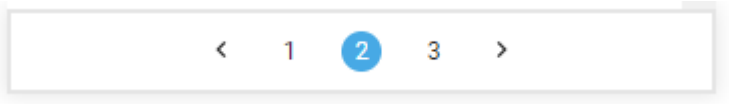


Description

The Table component displays database data in tabular form. Properties enable you to customize the data content, style, navigation, and user interaction of your table.

User Interaction

Interaction	Description
Column Resizing	When configured through the designer via the corresponding column config, a column can be resized during runtime. The resize handle exists in a 36px swath centered on the end of the header cell. Hovering over this area will change the mouse cursor to column resizing. Dragging the resize handle will display a resize guide effectively providing a visual for the new column position as the user drags. These changes in width will not persist, and are merely for the convenience of the user.
Sorting	When sorting is enabled on a column and the table head is enabled, a sort indicator will display to the right of the header cell content. The sort indicator will display the sort direction. • Single Sort - Enabled by double clicking on a header cell. • Multi Sort - Enabled by holding down Shift then double-clicking on multiple header cells. The Table component sorts columns based on the underlying data type in the column, instead of sorting alphanumerically as if all values were string formatted.
Selection	When selection is enabled, a user may select table data based upon the table's selection configuration. In the browser, selection is indicated by a light blue overlay rendered on cells. The root selection, or most recently selected cell has a light blue border. The root selection corresponds with the selected column and selected row properties of the table component's selection configuration. • Single - Single mouse click enabled. • Single Interval - Shift and single mouse click enabled. • Multiple Interval - Command/Ctrl + shift and single mouse click enabled.
Editing	When editing is enabled on an individual cell, a user can edit a cell by performing the interaction specified by the `allow edit on` property of the table component. When in edit mode, an editing cell with the corresponding cells content will be presented for edit. To commit this edit, the user must

Interaction	Description
	press the return or enter key. To exit the edit, the user may either press the escape key or select another table cell. When an edit is committed, the edited data is sent to the cell edit component event of the Table component. If an external source updates a cell value, which will cause either the `selectedRow` or `selectedColumn` property to change, the Table will page and scroll to the updated cell for increased visibility to Table edits.
Paging	When paging is enabled, a user may use the provided buttons to navigate between available pages and also jump to a specific page within range. 
Filtering	When filtering is enabled, a user may filter all of the data, not just the data being displayed when pagers are enabled, of the table component. If paging also happens to be enabled, the table will automatically page jump if it becomes necessary so that it does not display an empty page.
Coloring/Look	The table is made up of various subareas (rows, cells, etc). To aid with styling the component, these subareas have dedicated style objects that can be used to change the look. Furthermore, some parts of the table's property model allow for more fine tuned control of the look. For example, changing the color of all the rows on table can be accomplished by setting a background color on the `rows.style` object. However, if you wanted to alternate colors on each row, you could instead look towards the `rows.striped.color` object, which allows you to pick colors for even and odd rows separately.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description
data	Can be a dataset, an array of arrays, or an array of objects. The preferred (recommended) data is either a dataset or an array of objects. Individual data items can be a string, a number, or an object with reserved keys. Object data items must have a **value** property. Optionally they can also hav properties to indicate the style for the object and whether it is editable. `city: { value: 'Folsom', editable: true, style: { backgroundColor: 'grey', classes: [] |'' } }`
virtualized	Enables virtualization of table rows, which is an optimization method that only shows a portion of the underlying data on the chart at a time. While enabled, the table will only be populated with a smaller subset of data just the visible rows, and a few rows above and below. The idea being the component will be populated with new records as the user scrolls down the listing, assuming there are enough records to necessitate a scrollbar. Enabling virtualization generally results in a performance gain in the sessior in cases where the data property is populated with a large amount of content, as the table will only have to "load" a small subset of content. The trade off is that the table will need to load records as the user scrolls, so scrolling quickly may not feel as "smooth" when compared to disabling virtualization.
selection	When Selection is configured, a user will be able to select table data based upon the table's selection configuration. Similar to Vision module, you can select single, single interval, and multiple interval selection modes. The current selection and selection data is written back to the table components property tree. With the exception of the selection data property, the selection properties are bidirectional, meaning that if you were to change th

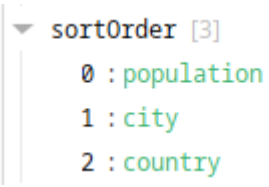
Name	Description
	value of the selected column property, it should be reflected in the table component. Click to see the **selection** properties.
filter	Where Table filtering is configured, as well as the filtered data. Click to see the **filter** properties.
enableHeader	When enabled, the table header is displayed including the main table heade along with the Header Groups. Default is true.
enableFooter	When selected, this enables the table footer, including the main table footer along with the Footer Groups. Default is false.
enableHeaderGroups	Enable table header groups if available. Default is false.
enableFooterGroups	Enable table footer groups if available. Default is false.
headerGroups	Header Groups are additional headers that are displayed above the main table header. Each header group equates to a single row with individual cel containing title text. The headerGroups properties include: • **title**: Text displayed in the header. Value is string. • **span**: You can use the span property to instruct individual cells to span multiple columns of the table. However, header group cells cannot spar more than the available columns. If you specify more columns in the span property than are actually available in the table, the cell will only span the remaining space. Value is numeric. • **justify**: Justify content horizontally. Options are left, right, and center. Value is string. • **align**: Aligns the content vertically. Options are top, center, or bottom. Value is string. • **style**: Sets a style that applies to header. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.

Name	Description
footerGroups	Footer Groups are additional footers that display above the main table footer. Each footer group equates to a single row which consists of individua cells containing title text. The footerGroups properties include: • **title**: Text displayed in the footer group. Value is string. • **span**: You can use the span property to instruct individual cells to span multiple columns of the table. However, footer group cells cannot span more than the available columns. If you specify more columns in the span property than are actually available in the table, the cell will only span the remaining space. Value is numeric. • **justify**: Justify content horizontally. Options are left, right, and center. Value is string. • **align**: Aligns the content vertically. Options are top, center, or bottom. Value is string. • **style**: Sets a style that applies to header. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.
columns	The Columns property allows for granular column-by-column configurations indicating how each column should be displayed in the table. Column configs enable the you to customize the table component's display and how users will be able to interact with the table in runtime. When a Column Config option is present, the table reflects that custom configuration. For examples on how column configurations work, see **Table Column Configurations**. You can add Column Config options by selecting **Add Array Element** unde the Columns property. By default, the Table component displays all availabl data in columns, however choosing to customize column configuration will reset the table to a single column display. Columns will have to be manually added back into the table using the **Add Array Element**  icon on the right of the Columns property. • **field**: A string that matches this column config with a table column. This string must correspond to the default column name of the column. Valu is string. • **visible**: Toggles column visibility. Allows table columns to be invisible to users, but data will be available to view params and selection. Value is

Name	Description
	boolean. • **editable**: Enables editing of all cells within this column. This can be overridden if the Editable property is set to false on an individual cell. Value is boolean. • **render**: The default render setting is auto. Can be auto, number, date, boolean, string, or view. When set to "view", the adjacent viewPath and viewParams properties can be used to specify which view, and set values on view parameters for the nested view. Value is string. • **justify**: Sets the justification for the content of the column. Options are left, center, right, or auto. The default setting is auto. Value is a string dropdown. • **align**: Sets the alignment for the content of the column. Options are top center, or bottom. The default alignment is center. Value is string. • **resizable**: Enables columns to be resized. When enabled, users can resize columns in the runtime by hovering over the edge of the column header then dragging the  cursor. Value is boolean. • **sortable**: Enables the column to be sorted. When enabled, users can double click on the column header in the run time to sort by ascending or descending order. Value is boolean. • **filter**: Column-specific filtering configuration. Click here to see the **columns.filter** properties. • **viewPath**: When render mode is set to View, the table will display the view found at the view path within each cell of this column. Value is string. • **viewParams**: Parameters to feed the configured view. Will be added to implicit parameters as follows: `{row:number;rowIndex:number;value:PlainObject;...viewParams}` • **boolean**: When render mode is set to Boolean, you can then specify how the boolean is represented in the runtime, for example, as a checkbox, toggle switch, value, and so forth. See Toggle Switch below. Value is string. • **number**: Type of component to render for boolean. Options are number or progress. When render mode is set to Number, you can then specify whether the number is represented in the runtime as value or as progress. Value is string.

Name	Description
	• **progressBar**: A progress bar configuration that is used when Number property is set to progress bar. You can specify the maximum value of the progress bar. Click here to see other **progressBar** properties. • **toggleSwitch**: Toggle switch configuration used when boolean is set to display as a toggle switch. Can specify selected and unselected color. ◦ **color.selected**: Color when the toggle switch is selected. See Color Selector. ◦ **color.unselected**: Color when the toggle switch is not selected. See Color Selector. • **numberFormat**: A number format string when render mode is set to number. Options are none, number [1,000.12], integer [1,200], four decimal precision [1.1200], percent [10.12%], scientific [1.01E+03], accounting [$(1,000.12)], financial [(1,000.12)], currency [$1,000.12], currency (rounded) [$1,012], duration [24:01:00], abbreviation [1.2k], or ordinal [100th]. You can also set any number of zeros to display the same amount of numbers after a decimal. For example, a [0,0.0] number format will display `1.1`, while a [0,0.00000] number format will display `1.12345`. Value is string. • **dateFormat**: Date format string used when render mode is set to date. Options are none, date [10/15/1018], time [3:59:00 PM], or date time [10/15/2018 15:59:00]. Value is string. • **width**: The width of this column. If resize is enabled, specifies the column width on initial load. User can override this in the runtime if the Resizable option is enabled. Value is numeric. • **strictWidth**: If enabled, the width of the column becomes fixed. Value i boolean. • **style**: Sets a style for this individual column. Full menu of **style option** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. This style is applied to the header, cells, and footer of the entire column. • **header**: Header cell configuration. ◦ **title**: Text for title of the column. Value is string. ◦ **justify**: Setting for justification of the title. Options are right, left, and center. Value is a string dropdown ◦ **align**: Setting for alignment of the title. Options are top, center, and bottom. Value is a string dropdown

Name	Description
	◦ **style**: Sets a style for this header. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **footer**: Footer cell configuration. ◦ **title**: Text for title of the footer. Value is string. ◦ **justify**: Setting for justification of the title. Options are right, left, and center. Value is a string dropdown ◦ **align**: Setting for alignment of the title. Options are top, center, and bottom. Value is a string dropdown ◦ **style**: Sets a style for this footer. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. **Implicit Properties** • **rowData**: Used to map parameters on a view cell to an entire row of data. The view must have a rowData object input parameter, with sub values that match the names of the columns. Then add the new view to the props.columns.0.viewPath property, and the input parameter as the props.columns.0.field property. Value is string.
dragOrderable	When enabled, users may drag column headers to reorder columns in the table if Column Config options are present.
sortOrder	The default weighted order in which columns and their contents are sorted relative to other columns and their contents. Used when the component loads. For **sortOrder** to be applied, the table must meet the following requirements: • Objects under the **columns** array must be defined for each column in the table's underlying data property you wish to display and sort on. ◦ In addition, each object under columns must have the **field** setting set to the data item under the data property (for example, "population" in the table's default data set). ◦ **sortable** must be enabled. ◦ **sort** must be set to something other than none. Once all columns have been configured, the sortOrder can be configured.

Name	Description
	Each element in the **sortOrder** array is expected to be a string value representing the name of the column (as determined by **field** value in the columns array). For example, sorting by population first, city second, and country last, would look like the following: 
rows	Configures all rows in the table component. Includes settings for expanding rows into subviews. Click to see the **rows properties**.
cells	Configures all cells in the table component. • **allowEditOn**: Enables the table cells to be edited on a single click, double click, or long press. Value is string. • **style**: Sets a style that applies to every cell in the table. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.
editingCell	The current editing cell with two properties to display the `column` and `row` when actively editing.
nullFormat	The table null format configuration used when a table contains either a "null string value or blank cell data. Can be overridden by individual column nullFormat configuration. • **includeNullStrings**:Toggles inclusion of "null" strings in null format logic. Default value is false. Value is boolean. • **strict**: Overrides render mode and apply nullFormatValue when enabled. Value is boolean. • **nullFormatValue**: Value to be applied against null values (or "null" strings if includeNullStrings is set to true), and includes three build-in options: blank, N/A, and null.
pager	Enables table pagination. Pagination improves performance and appearance on large tables, over 1000 rows. Click to see the **pager properties**.

Name	Description
resizeMode	Specifies whether the table resize mode is either Fill or Fixed. In Fill resized mode, the total width of all the columns cannot be less than the width of the table. In Fixed resized mode, the total width of all the columns can be less than the width of the table.
style	Sets a style that applies to the component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.
emptyMessage	Empty message configuration. • noData: Empty message configuration for when there is either no data source or the data source is empty. Click to see the **noData properties** • noFilterResults: Empty message configuration for when a filter returns no results. Click to see the **noFilterResults properties**. • style: Sets a style that applies to the empty message display area. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **styl class**.
headerStyle	Sets a style that applies to all column headers. Can be overridden by both columns.style and columns.header.style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.
headerGroupStyle	Sets a style that applies to all headerGroups. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.
bodyStyle	Sets a style that applies to the table body. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.
footerStyle	Sets a style that applies to all column footers. Can be overridden by both columns.style and columns.footer.style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.

Name	Description
footerGroupStyle	Sets a style that applies to all footerGroups. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.

selection Properties

Name	Description	Property Type
mode	This option determines if only one row, cell, or column can be selected at once. Options are single, single interval, and multiple interval.	value: string
enableRowSelection	This option is used in conjunction with the Column Selection allowed flag in order to determine whether whole rows, whole columns, or both (single cells) are selectable. Can be set to true or false. Default is true.	value: boolean
enableColumnSelection	This option is used in conjunction with the Row Selection allowed flag in order to determine whether whole rows, whole columns, or both (single cells) are selectable . Can be set to true or false. Default is false.	value: boolean
selectedColumn	The index of the first selected column, or null if none.	value: numeric
selectedRow	The index of the first selected row, or null if none.	value: numeric
data	An array of objects representing the current selection.	array
style	Sets a style that applies to the visual selection. This does not impact the highlight style when the mouse is hovered over the table. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

filter Properties

Name	Description	Property Type
enabled	Enables filtering. Default is false.	value: boolean
text	Contains the text you want to filter on.	value: string
results	The filtered data. • **enabled**: Enables the filter results to be written back to the props. Doing so may cause performance decline. Default is false. Value is boolean. • **data**: An array of objects representing the current filtered data if filtering is enabled. Each object represents a row of the table.	object
style	Sets a style that applies to the filter display. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

columns.filter Properties

Name	Description	Property Type
enabled	When true, will apply any valid column filters configured here.	boolean
visible	Specifies how the filter icon in the column header is visible to the user. Unless "never", will always be shown if a mobile device is connected.	value: string dropdown
string	String type column filter. • **condition**: The conditions by which the string filter will apply. Possible values include: ◦ contains ◦ equals ◦ starts with ◦ ends with • **value**: The specific string value that will be used for the filter. Value is string.	object
number	Number type column filter. • **condition**: The conditions by which the number filter will apply. Possible values include: ◦ greater than ◦ greater than or equal to ◦ less than ◦ less than or equal to ◦ equals ◦ between • **value**: The specific number value that will be used for the filter. Value is string.	object
boolean	Boolean type column filter. **condition**: The condition by which the boolean filter will apply. Possible values are true and false.	object

Name	Description	Property Type
date	Date type column filter. • **condition**: The condition by which the date filter will apply. Possible values include: ◦ contains ◦ date equals ◦ date time equals ◦ before date ◦ before date time ◦ before or equal to date ◦ before or equal to date time ◦ after date ◦ after date time ◦ after or equal to date ◦ after or equal to date time ◦ between dates ◦ between date times • **value**: The specific date time that will be used for the filter. Value is string.	object

columns.progressBar Properties

Name	Description	Property Type
max	Progress bar maximum value.	value: numeric
bar	Settings for the bar. • **color**: Color of the bar and the track. See [Color Selector]. • **style**: Sets a style for the progress bar. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
track	Settings for the track. • **color**: Color of the bar and the track. See [Color Selector]. • **style**: Sets a style for the progress bar. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
value	Settings for the value on the Progress Bar. • **enabled**: Whether or not to show the value. Value is boolean. • **format**: Format to apply to the value. Value is string. • **justify**: Horizontal alignment of the value. Value is string. • **style**: Sets a style for the value. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

rows Properties

Name	Description	Property Type
height	A minimum height value applied to all rows. A row cannot be shorter than this value, but it can be taller based on the height of the content it displays. This property can be set to "auto" (the default value) or given a numerical value that will correspond to a minimum row height in pixels.	value: string / numeric
subviewExpansionMode	Specifies how many subviews can be expanded at any given time. Options are multiple or single. Default is multiple.	value: string
subview	When enabled, each table row can be expanded into a subview. The Expandable Arrow ▼ opens the subview. Content of the subview is determined by the View Path property. See the **Displaying a Subview in a Table Row** page for an example. • **enabled**: Enable each row to allow toggling of the specified view. Value is boolean. • **viewPath**: A viewpath used to display a view as an expanded row. Value is string. • **viewParams**: Parameters to feed the configured view. Parameters specified here will be passed to the root of the "params" category of properties on the sub view. In addition to the properties above, subviews will be passed implicit parameters provided by the row: • **row**: A number representing the row • **rowIndex**: A number representing the index of the current row • **value**: JSON Object representing the contents of the table. The value of each column in the table will be a value under this object. Example: value.population.	

Name	Description	Property Type
striped	Settings for setting the striping (alternating background color) to the rows of the table. • **enabled**: When enabled (true), the table will be displayed with alternating background color to the rows of the table. Value is boolean. • **color**: Color settings: ◦ **even**: Background color for the even rows. See **Color Selector**. ◦ **odd**: Background color for the odd rows. See **Color Selector**.	object
highlight	Highlight settings. • **enabled**: When enabled (true), this feature will highlight the row that is currently selected or moused over. Value is boolean. • **color**: Highlight color for the row. See **Color Selector**.	object
style	Sets a style that applies to every row in the table. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

pager Properties

Name	Description	Property Type
options	Rows to show per pager option.	array
activeOption	Determines how many rows the table displays when first rendered, and can modify the number of rows displayed at runtime when bound to. Must exist as an available option.	value: numeric
top	Enables top pager. The pager is a menu that displays the current page and **Previous** ‹ and **Next** › icons for navigation.	value: boolean
bottom	Enables bottom pager. The pager is a menu that displays the current page and **Previous** ‹ and **Next** › icons for navigation.	value: boolean
activePage	Represents the current active page and corresponds to the value of the page jump input field.	value: numeric
style	Sets a style that applies to the pager container. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

emptyMessage.noData Properties

Name	Description	Property Type
text	Text to display when there is no data source or the data source is empty.	value: string
textStyle	Sets a style that applies to the text. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
icon	Settings for the icon to be displayed when there is no data source or the data source is empty. • **path**: Shorthand path to the icon source, in format: library/iconName. Value is string. • **color**: Color of the icon. Alternatively, you can use fill settings in the style property. Value is string. • **style**: Sets a style that applies to the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
bottom	Enables bottom pager. The pager is a menu that displays the current page and Previous ‹ and Next › icons for navigation.	value: boolean
activePage	Represents the current active page and corresponds to the value of the page jump input field.	value: numeric

emptyMessage.noFilterResults Properties

Name	Description	Property Type
text	Text to display when a filter returns no results.	value: string
textStyle	Sets a style that applies to the text. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
icon	Settings for the icon to be displayed when a filter returns no results. • **path**: Shorthand path to the icon source, in format: library/iconName. Value is string. • **color**: Color of the icon. Alternatively, you can use fill settings in the style property. Value is string. • **style**: Sets a style that applies to the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Scripting

See the **Perspective - Table Scripting** page for the full list of scripting functions available for this component.

Examples

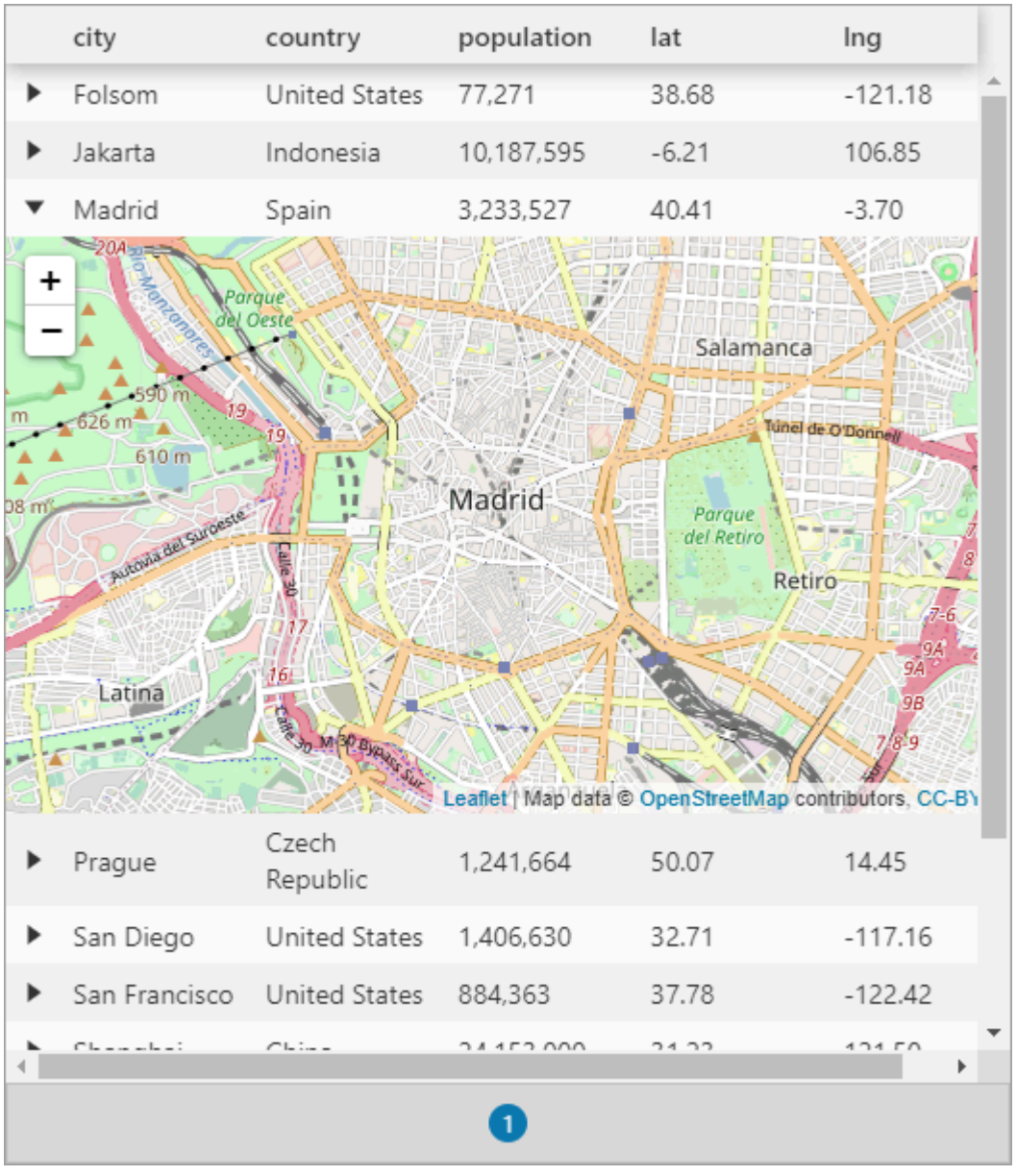
For examples of the Table component, see the following pages:

- **Displaying a Subview in a Table Row**
- **Styling Rows Based on Values**
- **Table Column Configurations**

Displaying a Subview in a Table Row

In a Perspective Table component, you have the option to enable subviews. When a subview is set up, you can click on the Expand icon and have another view display without closing the first view. This example sets up a table with several city statistics and when the Expand icon is selected for a city, a Map component will be displayed showing the location of the city on the map.

This example focuses on using a Map component in a subview on the table, but the larger implication here is that subviews in table rows can receive values from each row and utilize them with property bindings, allowing each subview to contain data unique to the row. The image below shows what our finished view will look like.



Summary of Subviews

When subviews are enabled for each row in a table, the row will implicitly pass some parameters over to the subview. The passed parameters are:

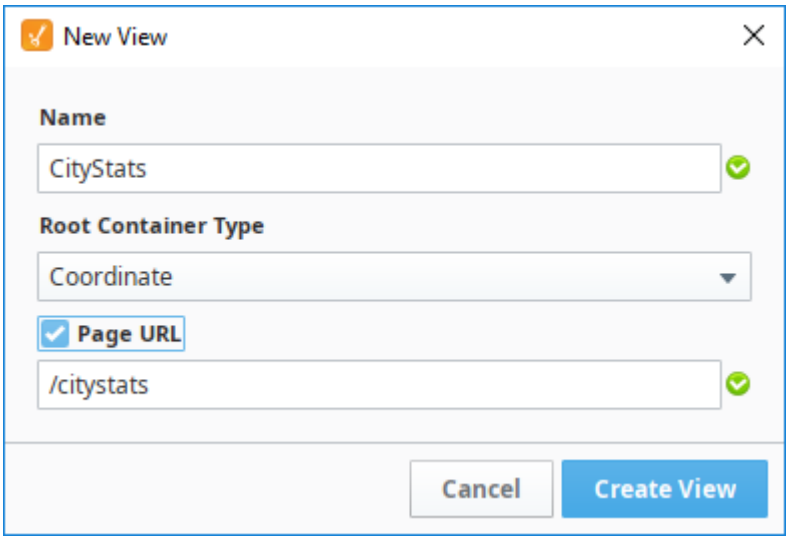
- row - a number representing the row
- rowIndex - a number reepresenting the index of the current row
- value - JSON Object representing the contents of the row. The value of each column in the row will be a value under this object.

Thus, in the image above where our table has a **country** column, the subview will receive the value of country at `params.value.country`. So you can create a component binding on the subview's configuration that references a value at `params.value.country`.

Create a View for the Table Data

We'll start by creating a view that will contain the table.

1. Right click on Views to create a view. In the example, we named the new view **CityStats**, set it as a **Coordinate** root container Type, and checked the Page URL option. Pages can be added later as well.



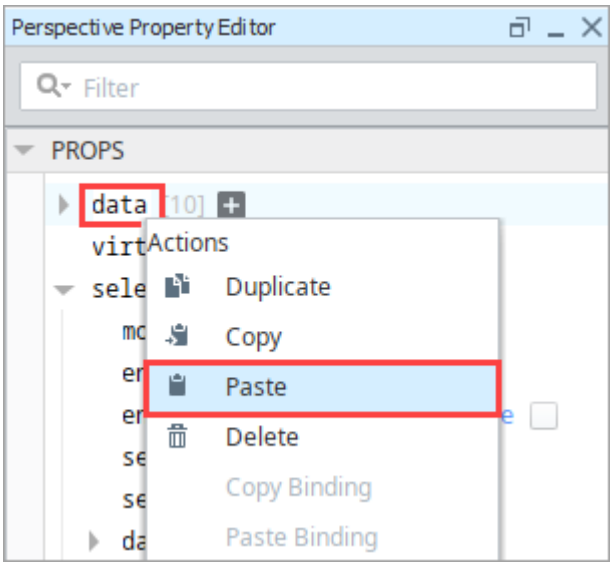
2. Drag a Table component onto the view.

3. The table needs to have Latitude and Longitude data for the map to show that location. Highlight and copy the following data:

```
[
{
    "city": "Folsom",
    "country": "United States",
    "population": 77271,
    "lat": 38.678287,
    "lng": -121.177318
},
{
    "city": "Jakarta",
    "country": "Indonesia",
    "population": 10187595,
    "lat": -6.208404,
    "lng": 106.849087
},
{
    "city": "Madrid",
    "country": "Spain",
    "population": 3233527,
    "lat": 40.41498,
    "lng": -3.702002
},
{
    "city": "Prague",
    "country": "Czech Republic",
    "population": 1241664,
    "lat": 50.073453,
    "lng": 14.450091
},
{
    "city": "San Diego",
    "country": "United States",
    "population": 1406630,
    "lat": 32.713832,
    "lng": -117.158616
},
{
    "city": "San Francisco",
    "country": "United States",
    "population": 884363,
    "lat": 37.776379,
    "lng": -122.423501
},
{
    "city": "Shanghai",
    "country": "China",
    "population": 24153000,
    "lat": 31.227167,
```

```
    "lng": 121.498839
},
{
    "city": "Tokyo",
    "country": "Japan",
    "population": 13617000,
    "lat": 35.69042,
    "lng": 139.746457
},
{
    "city": "Washington, DC",
    "country": "United States",
    "population": 658893,
    "lat": 38.90598,
    "lng": -77.04882
},
{
    "city": "Wellington",
    "country": "New Zealand",
    "population": 405000,
    "lat": -41.284336,
    "lng": 174.770488
}
]
```

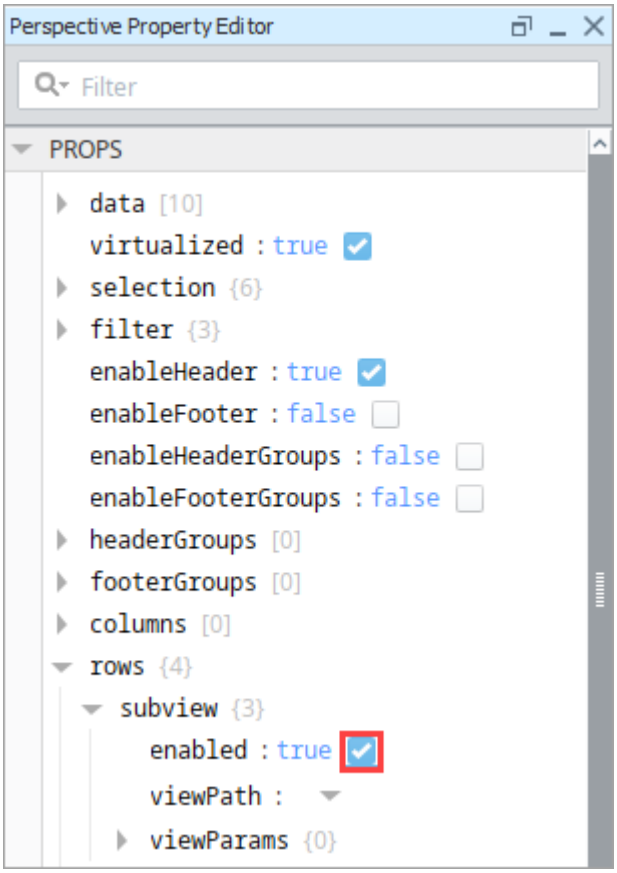
4. Right click on the data property of the Table component and select **Paste**.



5. Your table will now display the data for 10 rows and 5 columns.



6. Next, enable the **subview** property under props.rows.



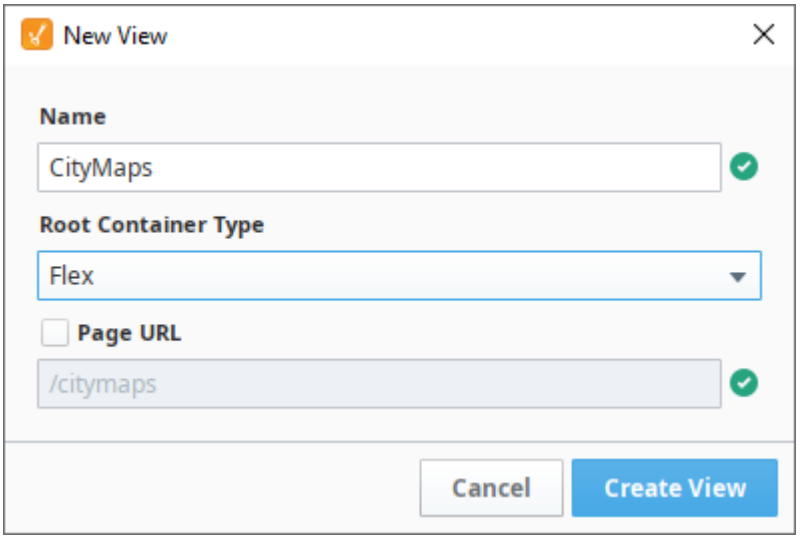
7. The table now has Expand ▶ icons for each row.

	city	country	population	lat	lng
▶	Folsom	United States	77,271	38.68	-121.18
▶	Jakarta	Indonesia	10,187,595	-6.21	106.85
▶	Madrid	Spain	3,233,527	40.41	-3.70
▶	Prague	Czech Republic	1,241,664	50.07	14.45
▶	San Diego	United States	1,406,630	37.71	-117.14

Create a View for Displaying the Map

Next we'll make the view that will be display a map of the cities in our table.

1. In the Project Browser, right click on Views to create a view. Name the new view **CityMaps**. Set it as a **Flex** layout, so the map easily takes up all available space. Lastly, do not check the Page URL option, as we don't need a corresponding page.



2. Drag a Map component onto the **CityMaps** View.

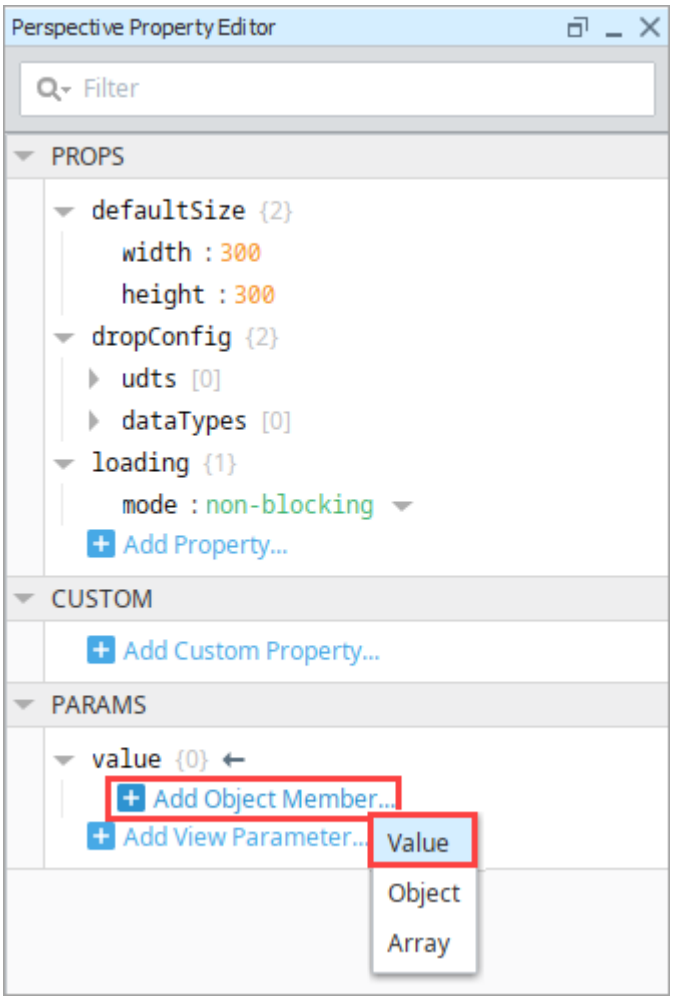
3. Set the Map's **position.grow** property to **1** so it resizes to take up the entire view.

4. Click on the **CityMaps** view in the Project Browser. In the Property Editor, click the **Add View Parameter** link under **Params** and choose **Object**.

5. Double click on key, and enter **value** as the object name. Note that the object **must** be named "value".

6. Next, we'll add two parameters to that value object.

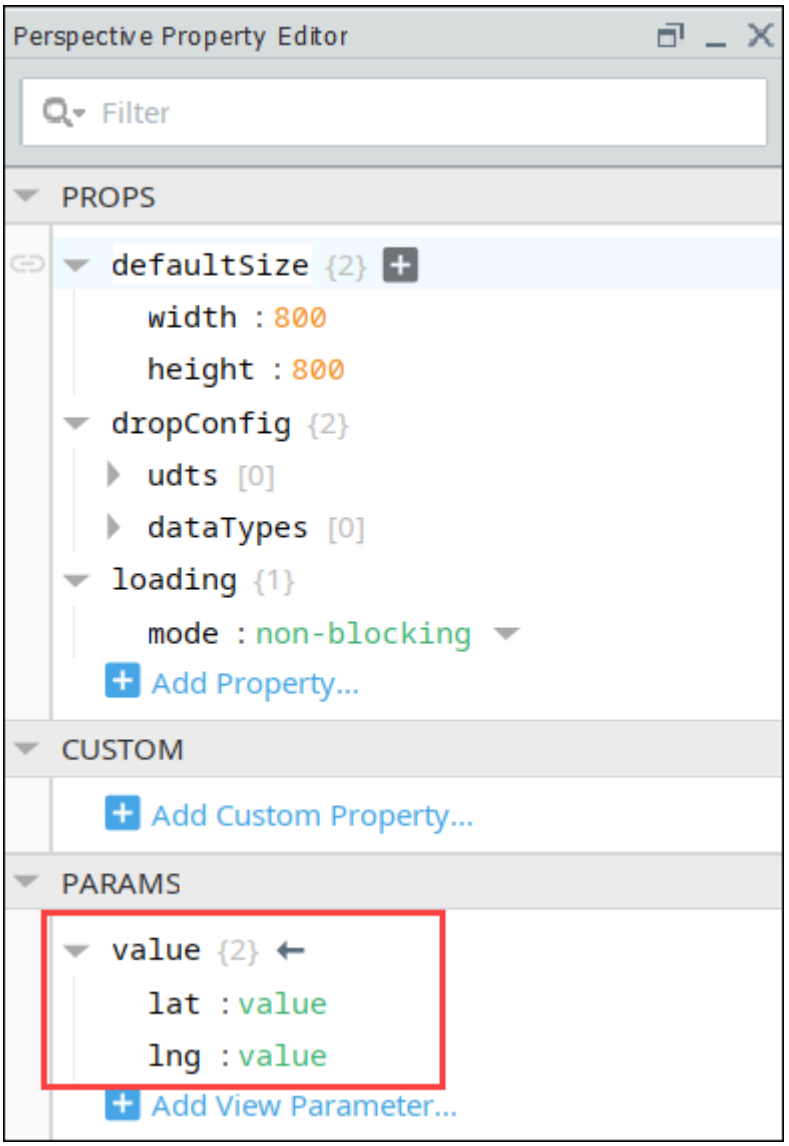
    i. Click **Add View Parameter** link under Params and choose **Value**.



ii. Double click on key, and enter "**lat**". This matches the lat (latitude) column from the Table on the CityStats view. This name must *exactly* match the column name in the table.

iii. Click the **Add Object Member** icon next to value and choose **Value**.

iv. Double click on key, and enter "**lng**". This matches the lng (longitude) column from the Table on the CityStats view. This name must exactly match the column name in the table.



> **NOTE**
>
> It's important that the object here is named "value". The perspective table we're going to embed the map into has a feature where content from each row in the table can be passed to its subview. This mechanism requires that the subview contain an object named "value". If you want the subview to be passed any of the values from its parent row, simply make a "value" object, and add values to it where the keys match the name of the column on the table.

7. Next select the Map component. We need to set the map's initial geographic center to the view parameters. In the Property Editor, expand the **init.center** property.

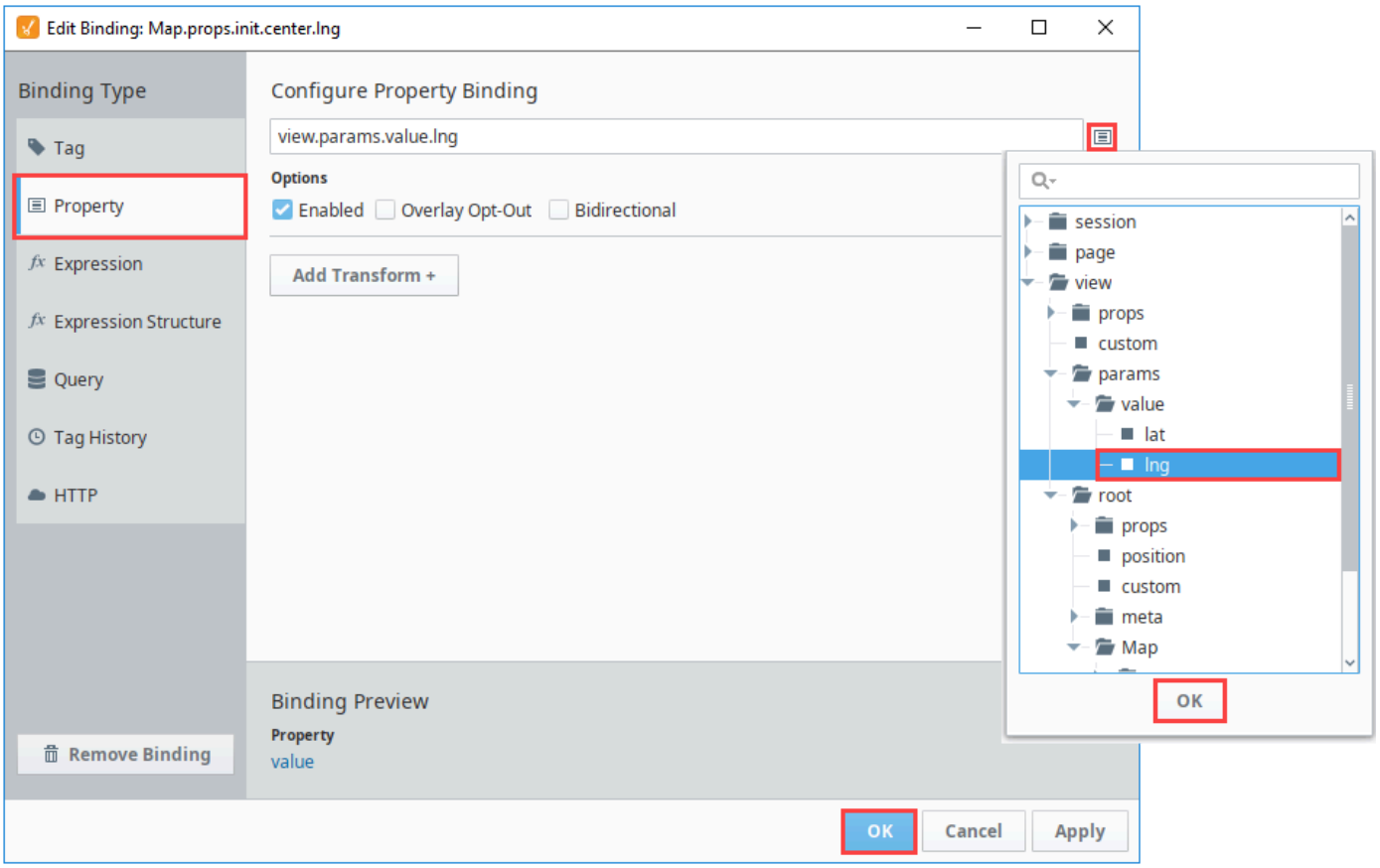
    i. Click on the **Binding** icon next to the **lat** property.

    ii. On the Edit Binding screen, select **Property** as the binding type.

    iii. Click the **Browse Properties** icon. Navigate to view, the params, the value, and then the **lat** property.

    iv. Click **OK**, then click **OK** again to save the binding.

> **NOTE**
>
> At this point, the init.center.lat property is bound to view.params.value.lat where view.params.value.lat's value is "value" instead of a valid latitude number. This will cause a Component Error which is expected.



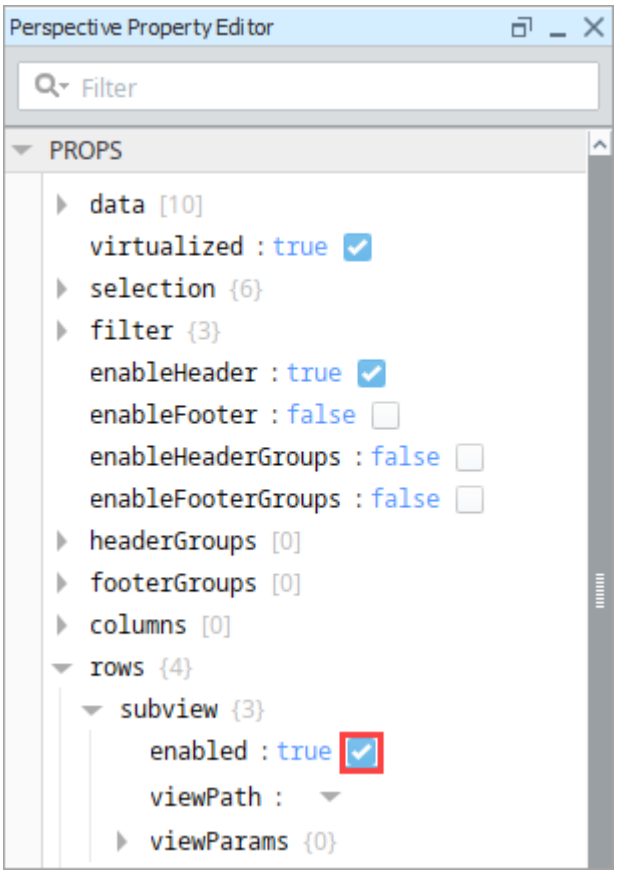
Use the Maps View as a Subview for the Table

Lastly, we need to tell the CityStats View to use CityMaps as its subview.

1. On the CityStats View, select the **Table** component.

2. In the Property Editor, scroll down to the **rows.subview.enabled** property.

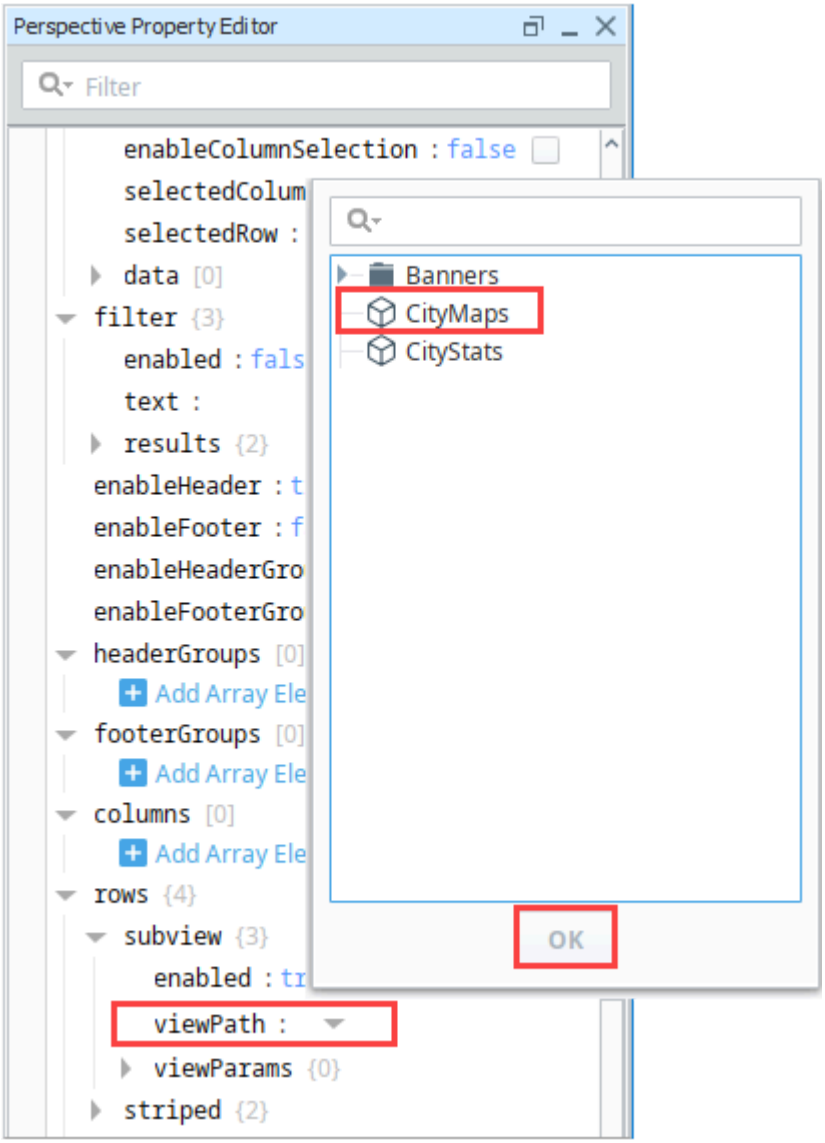
3. Next, enable the **enabled** property.



As a result, you'll notice the table now has **Expand** ▶ icons for each row.

	city	country	population	lat	lng
▶	Folsom	United States	77,271	38.68	-121.18
▶	Jakarta	Indonesia	10,187,595	-6.21	106.85
▶	Madrid	Spain	3,233,527	40.41	-3.70
▶	Prague	Czech Republic	1,241,664	50.07	14.45
▶	San Diego	United States	1,406,630	37.71	-117.14

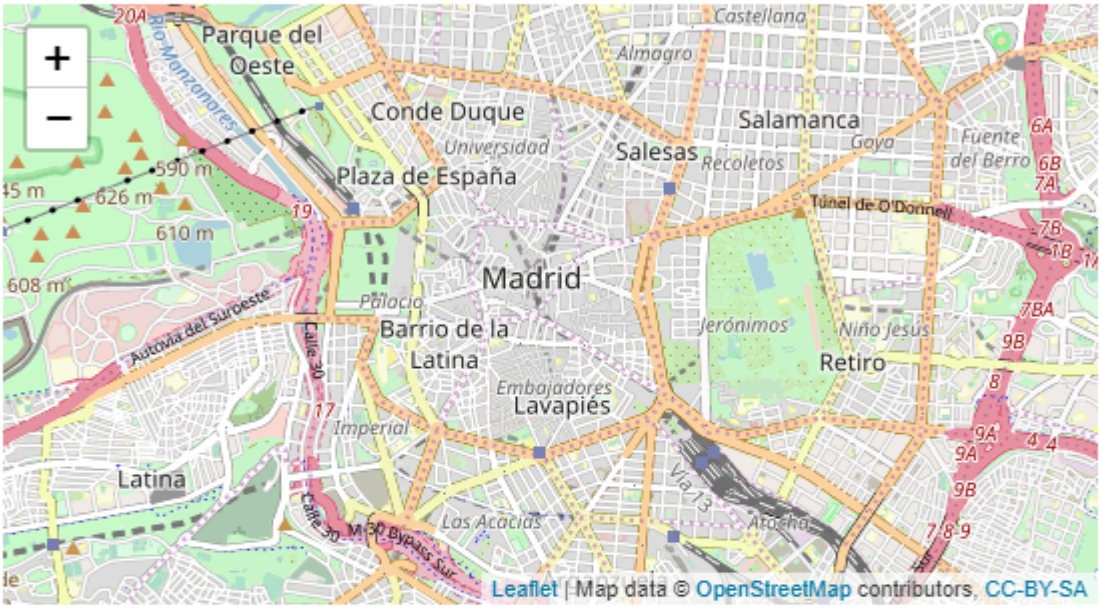
4. Find **rows.subview.viewPath**, and click the dropdown to see the a list of possible views. Choose **CityMaps** from the list and click **OK**.



5. **Save** your project.

6. Put the Designer into **Preview** mode. Click on the **Expand** ▶ icon next to one of the cities. You'll see a map of the city appear underneath the table row for that city. To close the map, click the **Collapse** ▲ icon.

	city	country	population	lat	lng
▶	Folsom	United States	77,271	38.68	-121.18
▶	Jakarta	Indonesia	10,187,595	-6.21	106.85
▼	Madrid	Spain	3,233,527	40.41	-3.70



▶	Prague	Czech Republic	1,241,664	50.07	14.45
▶	San Diego	United States	1,406,630	37.71	-117.14
▶	San Francisco	United States	884,363	37.78	-122.42
▶	Shanghai	China	24,153,000	31.23	121.50
▶	Tokyo	Japan	13,617,000	35.69	139.75
▼	Washington, DC	United States	658,893	38.91	-77.05

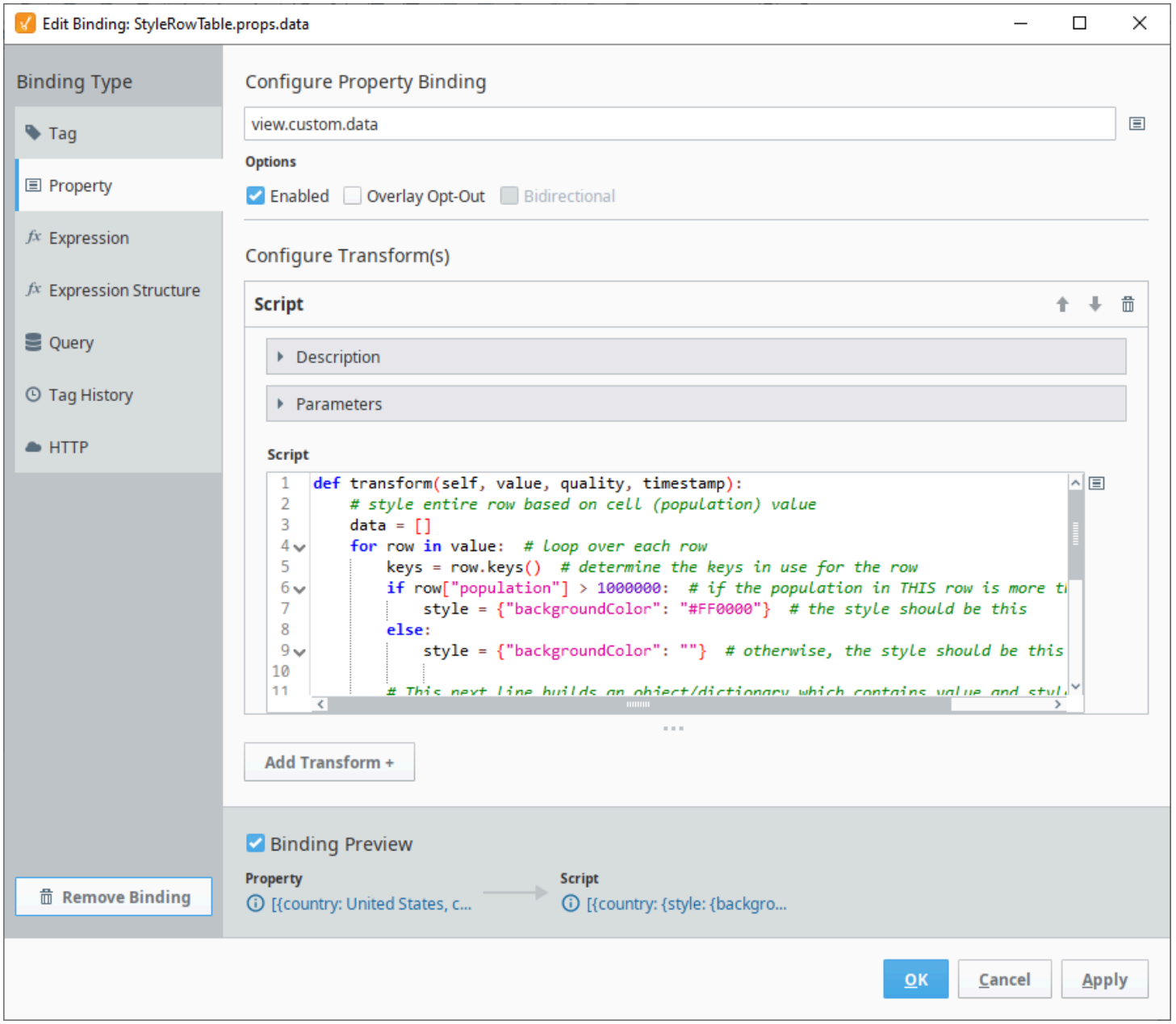
1

Styling Rows Based on Values

This page includes examples detailing different ways to style rows based on values using Script Transforms. Each example accomplishes similiar results through different binding and data types to change the background color of a row or individual cell.

Example 1: Styling Rows - Python Array/List

If the `data` property on the Table component has a Property binding configured to the `view.custom.data` property, we can add a Script Transform to specify a style condition for an entire row. The following example demonstrates this by changing the background color of rows that meet the value requirement to red.



The following code was used in this example to add the styling changes:

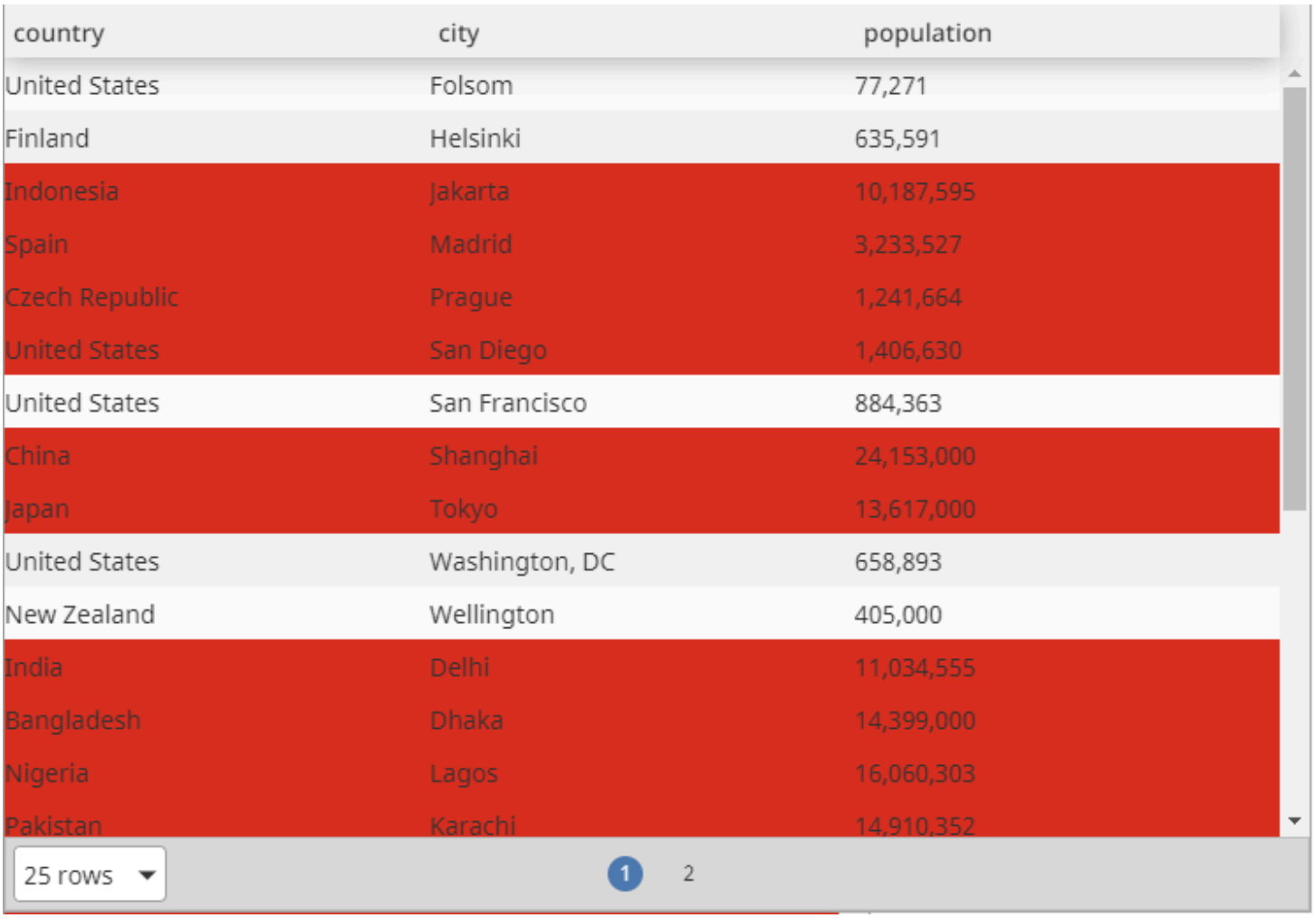
Style Entire Row Based on Cell Population Value

```
def transform(self, value, quality, timestamp):
        data = []
        for row in value:  # Loop over each row.
                keys = row.keys()  # Determine the keys in use for the row.
                if row["population"] > 1000000:  # If the population in THIS row
is more than 1000000...
                        style = {"backgroundColor": "#FF0000"}  # the style should
be this.
                else:
                        style = {"backgroundColor": ""}  # Otherwise, the style
should be this.

                # This next line builds an object/dictionary which contains value
and style keys for
                # every cell in the row. The value key will be used to store the
true value of the
                # object, while the style key will be used to define the
appearance of the CELL.
                # Since our goal is to style all cells in this row in the same
manner, we are re-using
                # the same style value across all cells in this row.
                formatted_rows = [{"value": v, "style": style} for v in
row.values()]

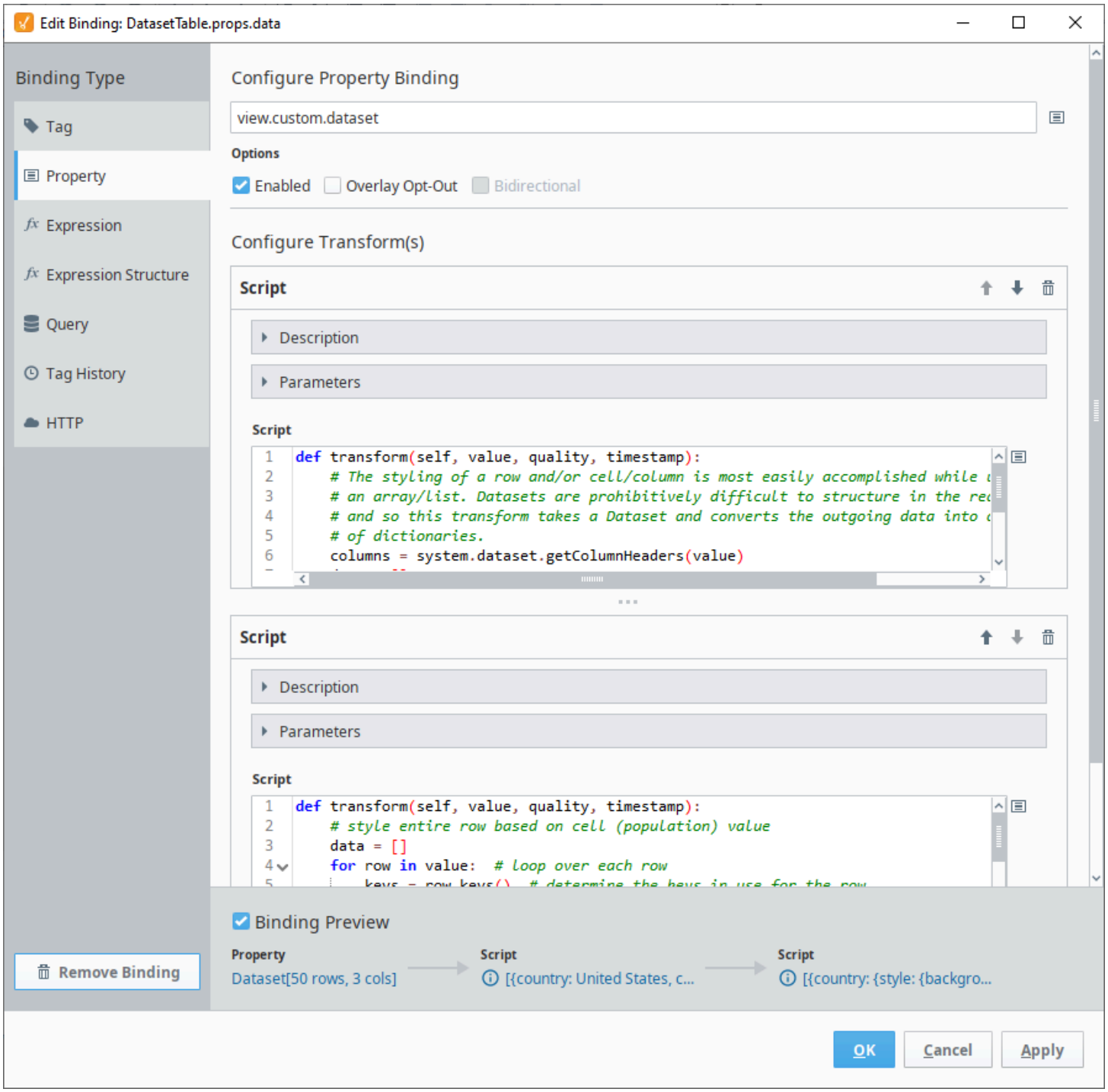
                # Here, we make a proper python dictionary out of the known keys
                # and our formatted cells.
                row_dict = dict(zip(keys, formatted_rows))
                data.append(row_dict)  # Now append this new dictionary to our new
data array/list.
        return data  # Finally, return our newly formatted data.
```

Clicking **OK** results in red backgrounds for entire rows where the Population column values are above 1,000,000.



Converting Datasets to an Array.

Styling a row or cell is best accomplished using an array/list. However, if your table values are provided through a dataset instead, you can configure your property binding to the `custom.dataset` property instead of the `custom.data` property. The following code is the second transform added to convert the dataset to an array/list before formatting the rows.



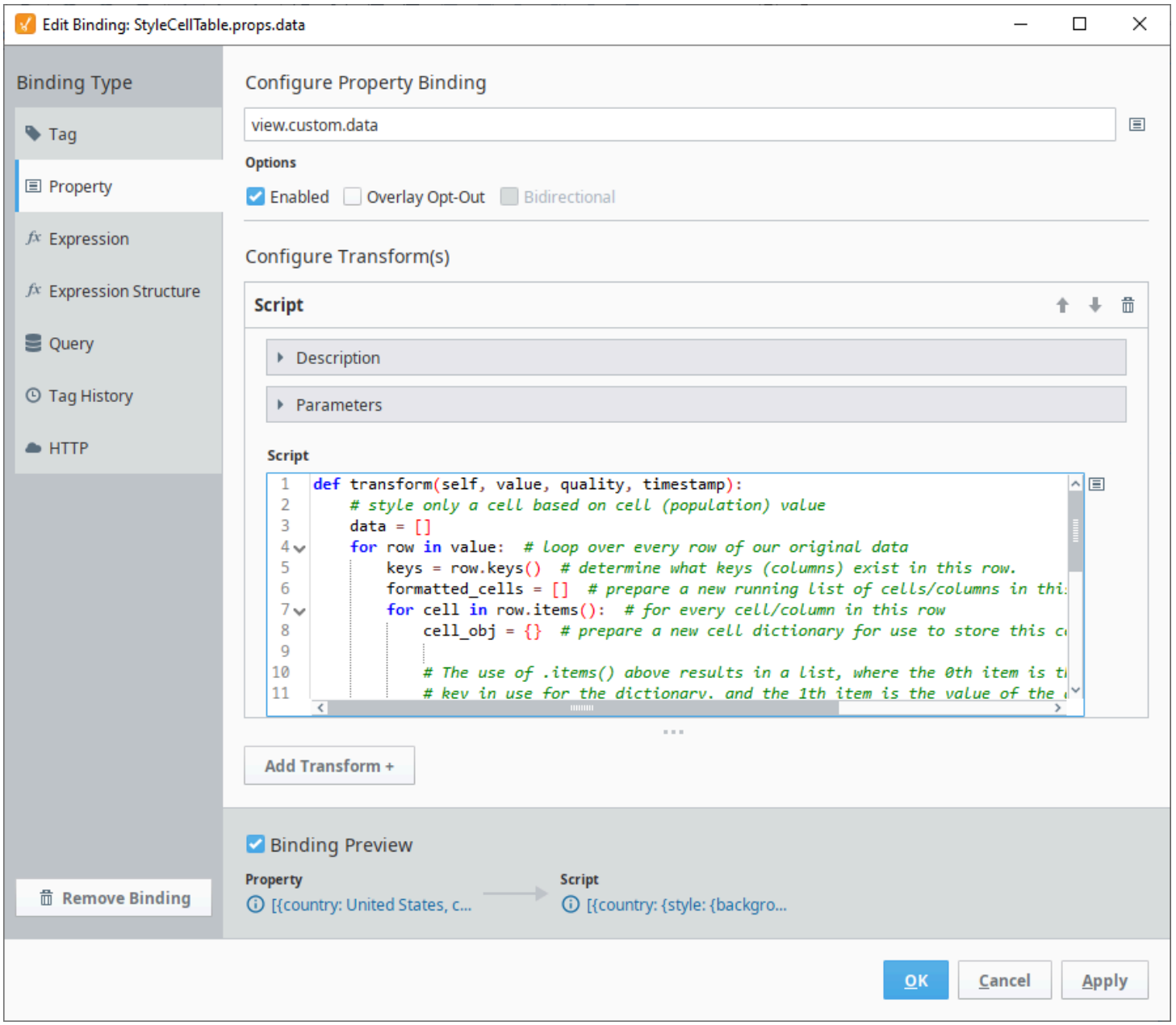
Convert Dataset into an Array/List of Dictionaries

```
def transform(self, value, quality, timestamp):
        columns = system.dataset.getColumnHeaders(value)
        data = []
        for row in range(value.getRowCount()):
                row_obj = {}
                for column in range(value.getColumnCount()):
                        row_obj[columns[column]] = value.getValueAt(row, column)
                data.append(row_obj)
        return data
```

Adding this Script Transform before the original Script Transform that styled the rows will result in the same row style results as the Table image above.

Example 2: Styling Singular Cells within Rows - Python Array/List

You can also style an individual cell instead of an entire row. For this example, we will still bind the `data` property on the Table component with a Property binding configured to the `view.custom.data` property, but the Script Transform will instead specify a style for individual cells in a row.



Style Only a Cell Based on Cell Population Value

```
def transform(self, value, quality, timestamp):
        data = []
        for row in value:  # Loop over every row of our original data.
                keys = row.keys()  # Determine what keys (columns) exist in this
row.
                formatted_cells = []  # Prepare a new running list of
cells/columns in this row.
                for cell in row.items():  # For every cell/column in this row
                        cell_obj = {}  # prepare a new cell dictionary for use to
store this cell.

                        # The use of .items() above results in a list, where the
0th item is the
                        # key in use for the dictionary, and the 1th item is the
value of the dictionary.
                        # We need to see if the key in use for this cell/column is
the key we are
                        # basing our styling off of, and if so we need to see if
the corresponding value
                        # exceeds the threshold specified to use the style. Note
here that the style is
                        # specified for each individual cell/column.
                        if cell[0] == "population" and cell[1] > 1000000:
                                style = {"backgroundColor": "#FF0000"}
                        else:
                                style = {"backgroundColor": ""}

                        cell_obj = {"value": cell[1], "style": style}  # Prepare a
new cell dictionary.
                        formatted_cells.append(cell_obj)  # Append our cell
dictionary to the running list of cells for this row.
                        row_obj = dict(zip(keys, formatted_cells))  # Make a
proper dictionary out of our formatted cells.
                data.append(row_obj)  # Append our row to our new data array/list.
        return data  # Return our new data array/list.
```

Clicking **OK** will now show individually styled cells in the Population column based on values above 1,000,000.

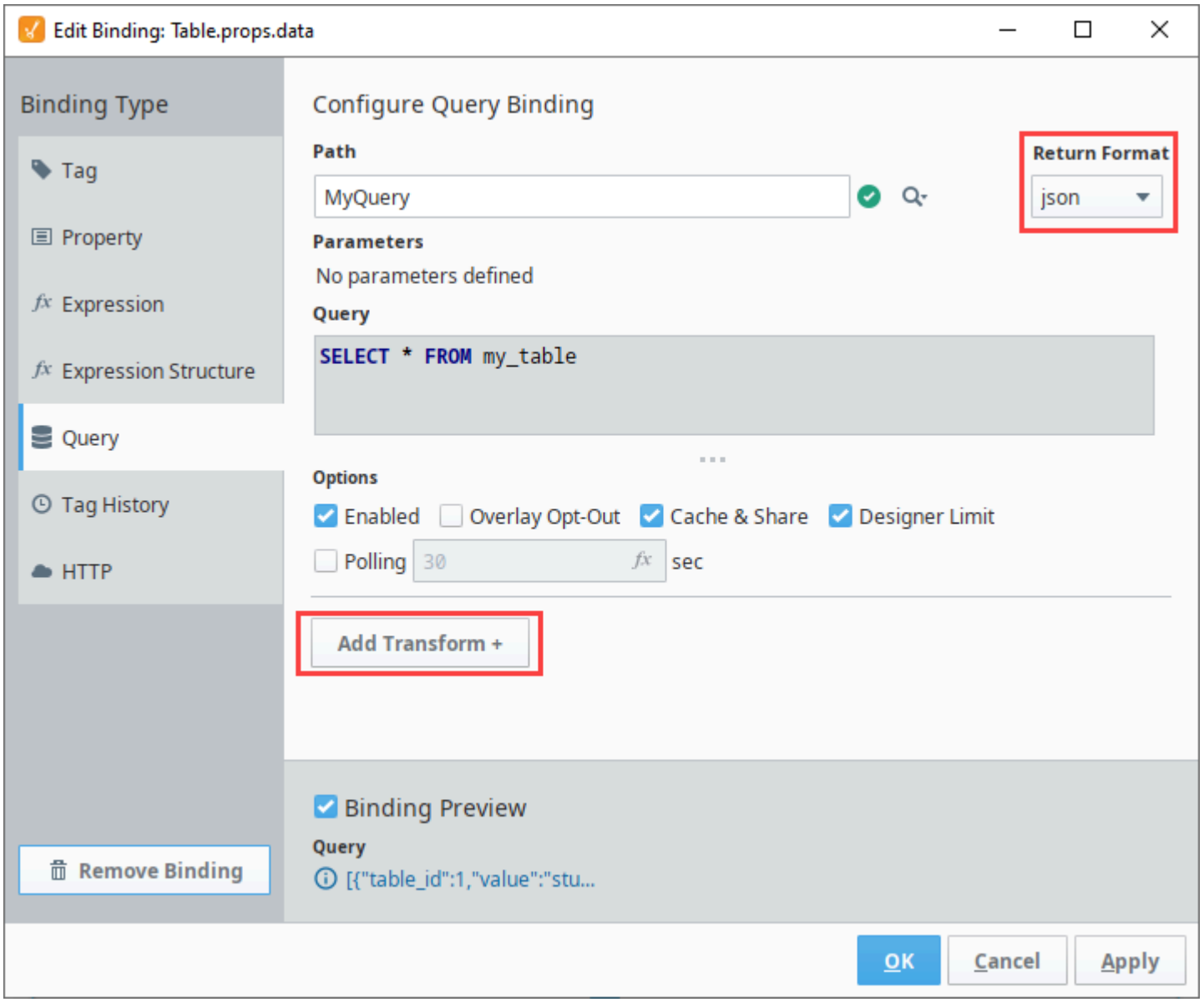
country	city	population
United States	Folsom	77,271
Finland	Helsinki	635,591
Indonesia	Jakarta	10,187,595
Spain	Madrid	3,233,527
Czech Republic	Prague	1,241,664
United States	San Diego	1,406,630
United States	San Francisco	884,363
China	Shanghai	24,153,000
Japan	Tokyo	13,617,000
United States	Washington, DC	658,893
New Zealand	Wellington	405,000
India	Delhi	11,034,555
Bangladesh	Dhaka	14,399,000

25 rows

1 2

Example 3: Styling Rows - JSON Data

If the `data` property on the Table component has a Query binding configured, we can set the Return Format to JSON, and then add a Script Transform:



In the Script Transform, we can add code that adds a styling to each value in the Named Query results:

Add Styling to each Value Returned by a Named Query

```
# Create a new list to store a modified result set from our query
newData = []

# Iterate over each row in 'value', which is the original result set
for row in value:

    # Within the row, iterate over each column
    for col in row:

        # Create a variable to store the contents of the original "cell"
        cell = row[col]

        # Create a dictionary containing the original value, and some styling
information
        row[col] = {"value": cell, "style": {"backgroundColor": "#00FF00"}}

    # Add the modified row to the list we initialized earlier
    newData.append(row)

#Return the list
return newData
```

Example 4: Styling Rows - Dataset Data

You can also change the color of rows in a table when the source of data is a dataset. This approach involves recreating the original data as a new JSON document that contains a style object for each row.

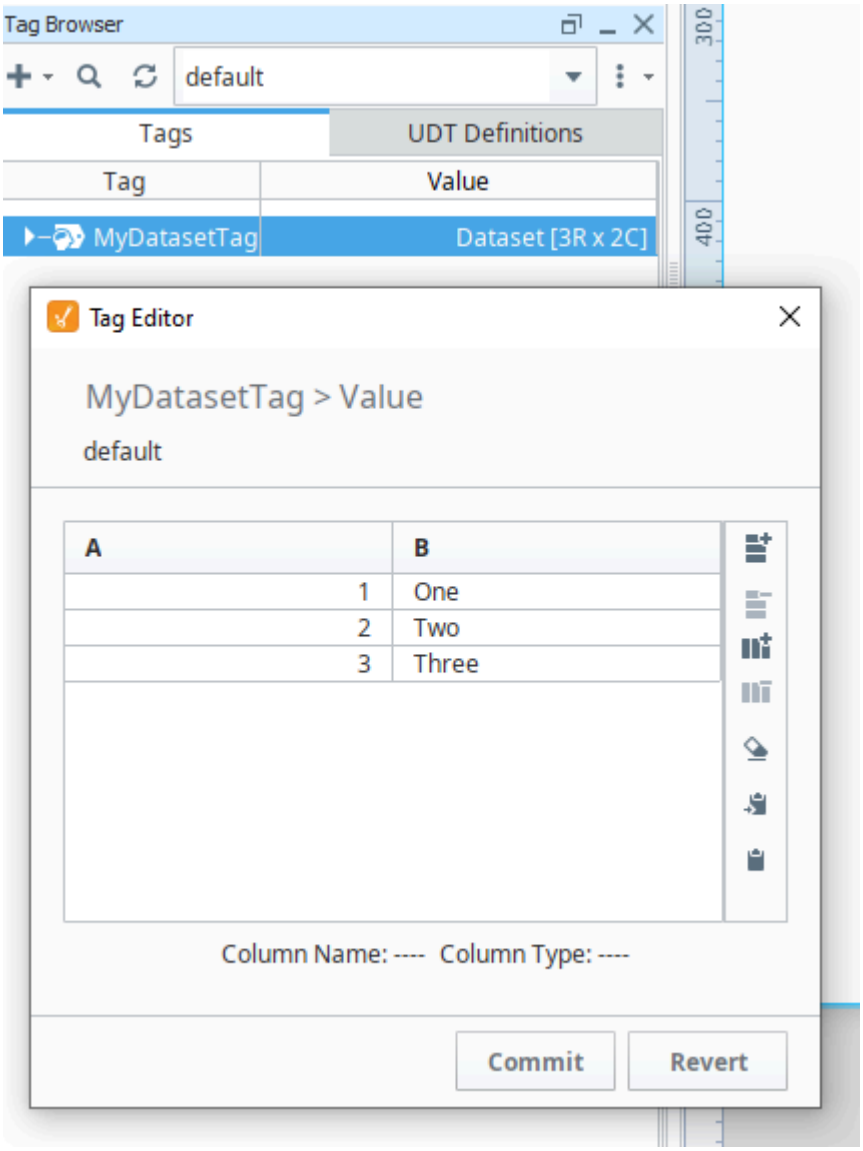
For example, say there's a memory tag set to a dataset type, which contains the following data:

A	B
1	One
2	Two
3	Three

Tag's JSON structure

```
```

```
{
  "valueSource": "memory",
  "dataType": "DataSet",
  "name": "MyDatasetTag",
  "value": "{\"columns\":[{\"name\":\"A\",\"type\":\"java.lang.Integer\"},
{\"name\":\"B\",\"type\":\"java.lang.String\"}],\"rows\":[[1,\"One\"],[2,\"Two\"],
[3,\"Three\"]]}",
  "tagType": "AtomicTag"
}
{
  "valueSource": "memory",
  "dataType": "DataSet",
  "name": "MyDatasetTag",
  "value": "{\"columns\":[{\"name\":\"A\",\"type\":\"java.lang.Integer\"},
{\"name\":\"B\",\"type\":\"java.lang.String\"}],\"rows\":[[1,\"One\"],[2,\"Two\"],
[3,\"Three\"]]}",
  "tagType": "AtomicTag"
}
```

We can bind the table component's `props.data` property to this tag with a Tag binding. From here we can add a Script Transform with the following:

```
# This list will be used to create a JSON like structure that will insert rows for
our styles
output_json = []

# Here we can define what styling on our rows will be.
style_orange = {"backgroundColor": "#F7901D"}
style_green = {"backgroundColor": "#00AA00"}

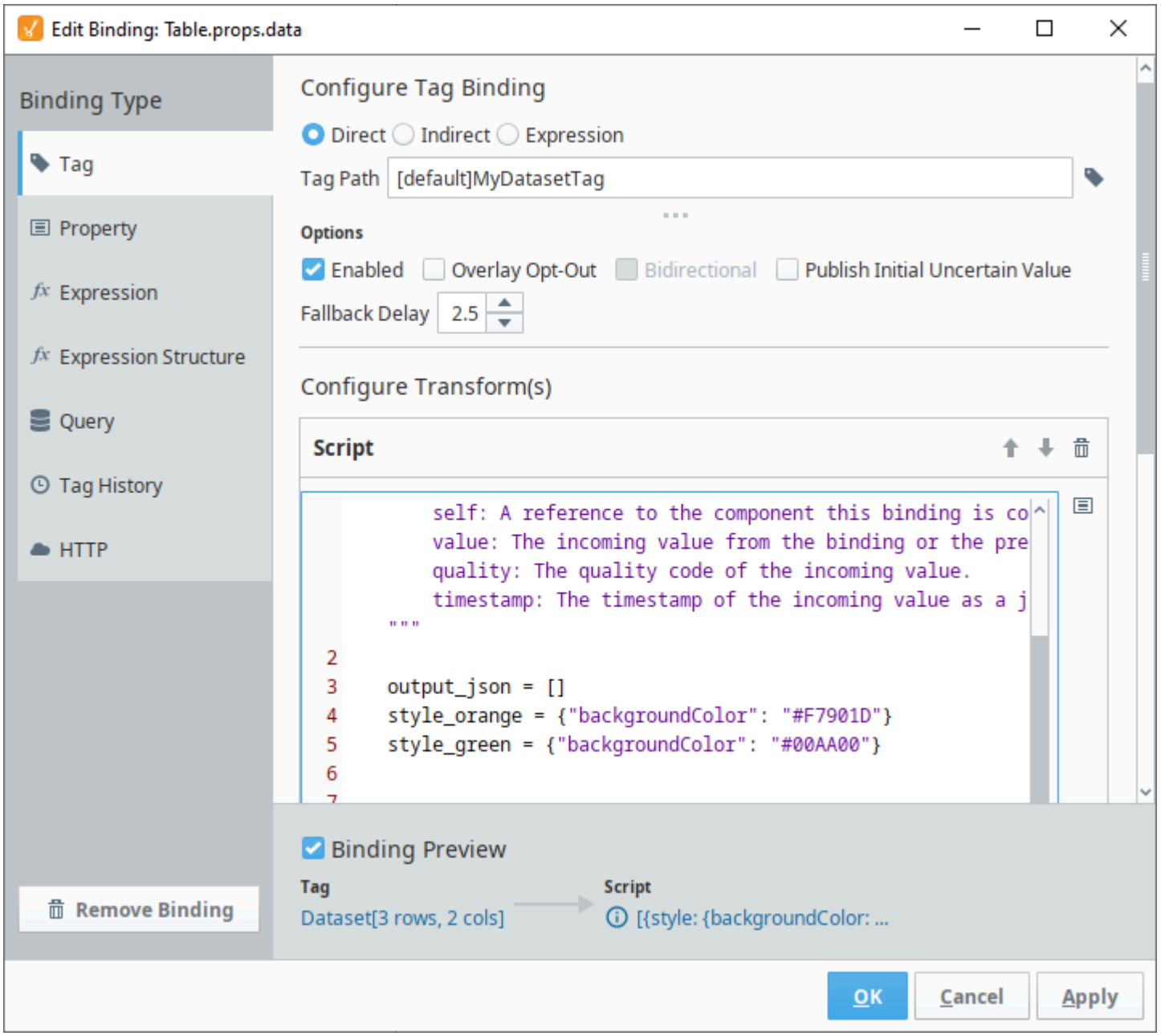
# You could change more than just the background color, for example:
# style_another_example {"backgroundColor": "#00AA00", "font-weight": "bold"}


for row in range(value.getRowCount()):
    row_object = {}
    row_value = {}
    row_style = {}
    for col in range(value.getColumnCount()):
        row_value[value.getColumnName(col)] = value.getValueAt(row, col)
        row_object['value'] = row_value

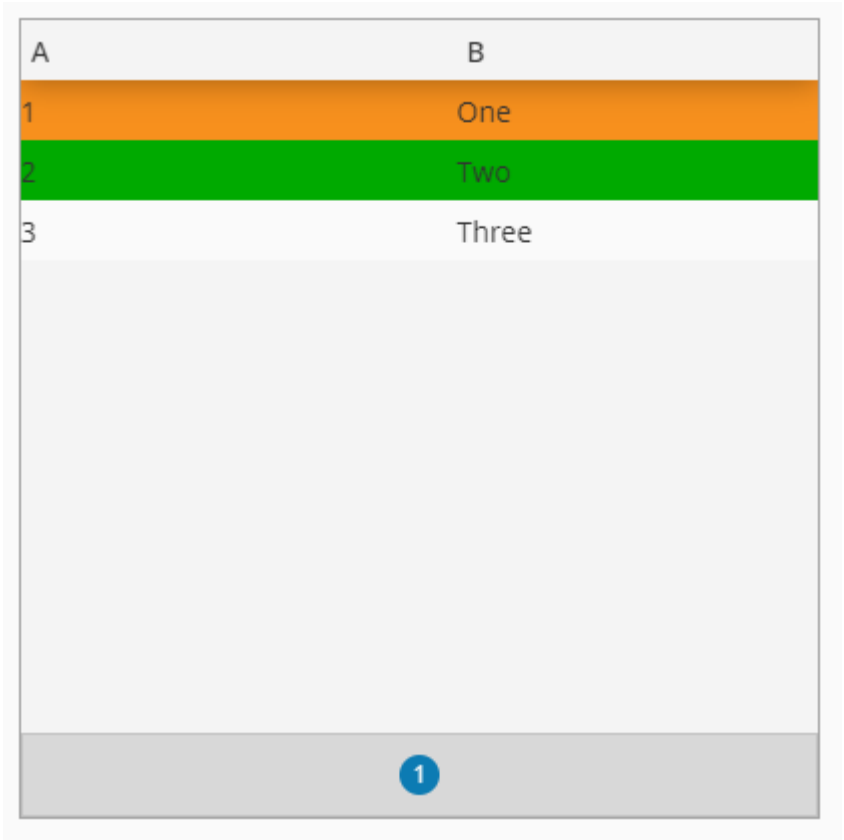
        # Here we're checking the name of the column that we want to base our
styling on.
        if value.getColumnName(col) == 'B':

            # Here we're checking for individual values within the column, and
applying styling
            if value.getValueAt(row, col) == 'One':
                row_style = style_orange
            elif value.getValueAt(row, col) == 'Two':
                row_style = style_green

        row_object['style'] = row_style
    output_json.append(row_object)
return output_json
```



This would result in coloring rows where the column B has distinct values of **One** or **Two**.


A
B
1
One
2
Two
3
Three
1

Perspective - Table Scripting

This page details the various scripting, component, and extension functions available for Perspective's **Table** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onEditCellCommit

This onEditCellCommit event is used with a runAction script on a table to take user entry and store it in the table or a database.

Provides a chance do something once a user has typed something into a cell. The user must commit the new value before the event will trigger. "Committing" a value depends on the type of value and how it's rendered. Numerical and text values can be committed by pressing "Enter" after typing a new value. Boolean values are typically committed via a click (such as in cases when the cell is rendered as a checkbox or toggle switches).

Additionally, the cell must first be editable (props.data.[rowNumber].[columnName].editable is set to true). The first cell in the default dataset on a newly created instance of the component demonstrates where the editable property must be positioned.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.column

- Object Path
    - event.column

- Type
  - String
- Description
  - The name of the column under which the cell was edited.

event.row

- Object Path
  - event.row
- Type
  - Number
- Description
  - The unique row index as it is represented in the source data. Also known as the row ID.

event.value

- Object Path
  - event.value
- Type
  - Any
- Description
  - The value that was typed into the cell.

event.originalValue

New in 8.3.3

- Object Path
  - event.originalValue
- Type
  - String

- Description
    - The original value before editing began.

Example - Change the value in a cell

```
# This example will set the value of a cell, based on what the user typed into it.

# Get the value that was typed into the cell
valueToSet = event.value

# We need to set a value in a particular cell. The event object contains row and
column properties
#  that report the position of the cell that was edited.

# If the data property contains an array, you would use the line below
self.props.data[event.row][event.column] = valueToSet

# If the data property contains a dataset, then you would want to use the
following line instead
#self.props.data = system.dataset.setValue(self.props.data, event.row,
event.column, valueToSet)
```

onSelectionChange

This onSelectionChange event will trigger when the selection in the chart changes.

> **NOTE**
>
> The onSelectionChange event will fire on startup or mount if props do not equal the table components default selection config.
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.selectedColumn

- Object Path
    - event.selectedColumn
- Type

    - String or null
- Description
    - The name of the column that the selected cell is located under.

event.selectedRow

- Object Path
    - event.selectedRow
- Type
    - Number or null
- Description
    - The unique row index as it is represented in the source data. Also known as the row ID.

event.data

- Object Path
    - event.data
- Type
    - Array
- Description
    - Represents the currently selected entries. The contents of the array is based on the enabledRowSelection and enableColumnSelection properties as represented on the table below. The actual resulting value may include additional values if the selection mode on the table is set to "single interval" or "multiple interval".

Properties	Resulting return type	Output
enabledRowSelection: True **enableColumnSelection**: False	An array containing a number of JSON objects that each represent a single row. Each JSON object contains one key-value pair for each column on the table.	`[{"city":"Folsom","country":"United States", "population":77271}]`
enabledRowSelection: False **enableColumnSelection**: True	An array of JSON objects, where each object represents a separate row in the selected column. Each object contains a single key-value	`[{"city":"Folsom"}, {"city":"Helsinki"}, {"city":"Jakarta"},]`

Properties	Resulting return type	Output
	pair, where the key is the column name and the value is the value of the cell.	
enabledRowSelection: True **enableColumnSelection**: True	An array containing a single JSON object, which can be treated like a Python dictionary.	`[{"city":"Folsom"}]`

onEditCellStart

This onEditCellStart event fires when the user starts editing a cell. For onEditCellStart, the value is the initial value before any edits.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.column

- Object Path
    - event.column

- Type
  - String or number
- Description
  - The column the editing cell is positioned under.

event.row

- Object Path
  - event.row
- Type
  - Number
- Description
  - The unique row index as it is represented in the source data. Also known as the row ID.

event.rowIndex

- Object Path
  - event.rowIndex
- Type
  - Number
- Description
  - The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

event.value

- Object Path
  - event.value
- Type
  - Any

- Description
    - The value of the cell before editing began.

onEditCellCancel

This onEditCellCancel event is fired when the user has canceled a cell edit and has exited editing mode by effectively pressing the escape key.

> **NOTE**
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.column

- Object Path
    - event.column
- Type
    - String
- Description
    - The column name of the cell being edited.

event.row

- Object Path
    - event.row
- Type
    - Number
- Description
    - The unique row index as it is represented in the source data. Also known as the row ID.

event.rowIndex

- Object Path
  - event.rowIndex
- Type
  - Number
- Description
  - The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

event.value

- Object Path
  - event.value
- Type
  - Any
- Description
  - The value of the cell before editing began.

event.canceledValue

New in 8.3.3

- Object Path
  - event.canceledValue
- Type
  - String
- Description
  - The value that was being edited when the edit was canceled.

onRowClick

This onRowClick event is fired when a row in the table is clicked.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.row

- Object Path
  - event.row
- Type
  - Number
- Description
  - The unique row index as it is represented in the source data. Also known as the row ID.

event.rowIndex

- Object Path
  - event.rowIndex
- Type
  - Number
- Description
  - The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

event.value

- Object Path
  - event.value

- Type
  - PlainObject
- Description
  - The rows value as a JSON object.

onRowDoubleClick

This onRowDoubleClick event is triggered when a row in the table is double clicked.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.row

- Object Path
  - event.row
- Type
  - Number
- Description
  - The unique row index as it is represented in the source data. Also known as the row ID.

event.rowIndex

- Object Path
  - event.rowIndex
- Type
  - Number
- Description

- The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

event.value

- Object Path
  - event.value
- Type
  - PlainObject
- Description
  - The rows value as a JSON object.

onSubviewExpand

This onSubviewExpand event is triggered when a row subview is expanded.

> **NOTE**
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.row

- Object Path
  - event.row
- Type
  - Number
- Description
  - The unique row index as it is represented in the source data. Also known as the row ID.

event.rowIndex

- Object Path

    - event.rowIndex
- Type
    - Number
- Description
    - The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

event.value

- Object Path
    - event.value
- Type
    - PlainObject
- Description
    - The rows value as a JSON object.

onSubviewCollapse

This onSubviewCollapse event is triggered when a row subview is collapsed.

> **NOTE**
>
> This component event is designed to be used in tandem with a script runAction. Within the script action, special properties and methods are available on the event object, which is passed to the script action as a parameter.

event.row

- Object Path
    - event.row
- Type
    - Number

- Description
    - The unique row index as it is represented in the source data. Also known as the row ID.

event.rowIndex

- Object Path
    - event.rowIndex
- Type
    - Number
- Description
    - The row index as it is represented in the current visible data. Useful in cases where some of the rows are hidden, such as when filtering.

event.value

- Object Path
    - event.value
- Type
    - PlainObject
- Description
    - The rows value as a JSON object.

Component Functions

.collapseSubviews()

- Description
    - This function will collapse the specified row subviews. If no parameter is specified, this function will collapse all expanded subviews on the current page.
- Parameters

    - array **rows** - An optional array of indices of rows to collapse. Any argument that is not a list will throw an exception. A list of invalid indices will not throw an exception. Omitting this parameter will collapse all subviews.

- Return
    - Nothing

> **NOTE**
>
> There is a distinction between using `row` and `rowIndex`. `Row` refers to the true index of the row as it exists in the data, and is not affected by paging, sorting, or searching. `RowIndex` refers to the visual index of the row as it appears on the table and is affected by paging, sorting, and searching.

Example

```
# Collapse subviews for rows 1 and 3 if they exist. If the list does not match any indices that exist, nothing will happen.
self.getSibling('Table').collapseSubviews([1, 3])

# Collapse all expanded subviews.
self.getSibling('Table').collapseSubviews()

# The following lines are invalid and will throw an exception:
self.getSibling('Table').collapseSubviews(None)
self.getSibling('Table').collapseSubviews(3)
```

.expandSubviews()

- Description
    - This function will expand the specified row subviews. This will only expand rows that are visible on the current page.

- Parameters
    - array **rows** - An array of indices of rows to expand. Any argument that is not a list will throw an exception. A list of invalid indices will not throw an exception. Omitting this parameter will expand all subviews.

- Return

- Nothing

> **NOTE**
>
> There is a distinction between using `row` and `rowIndex`. `Row` refers to the true index of the row as it exists in the data, and is not affected by paging, sorting, or searching. `RowIndex` refers to the visual index of the row as it appears on the table and is affected by paging, sorting, and searching.

Example

```
# Expand subviews for rows 1 and 3 if they exist. If the list does not match any
indices that exist, nothing will happen.
self.getSibling('Table').expandSubviews([1, 3])

# Expand all subviews.
self.getSibling('Table').expandSubviews()

# The following lines are invalid and will throw an exception:
self.getSibling('Table').expandSubviews(None)
self.getSibling('Table').expandSubviews(3)
```

Extension Functions

This component does not have extension functions associated with it.

Table Column Configurations

This page contains examples that involve making changes to a table component's columns property.

Aligning Text in a Column

The table component can apply different text alignment for each column. In this example we will demonstrate the concept with the default dataset on the component.

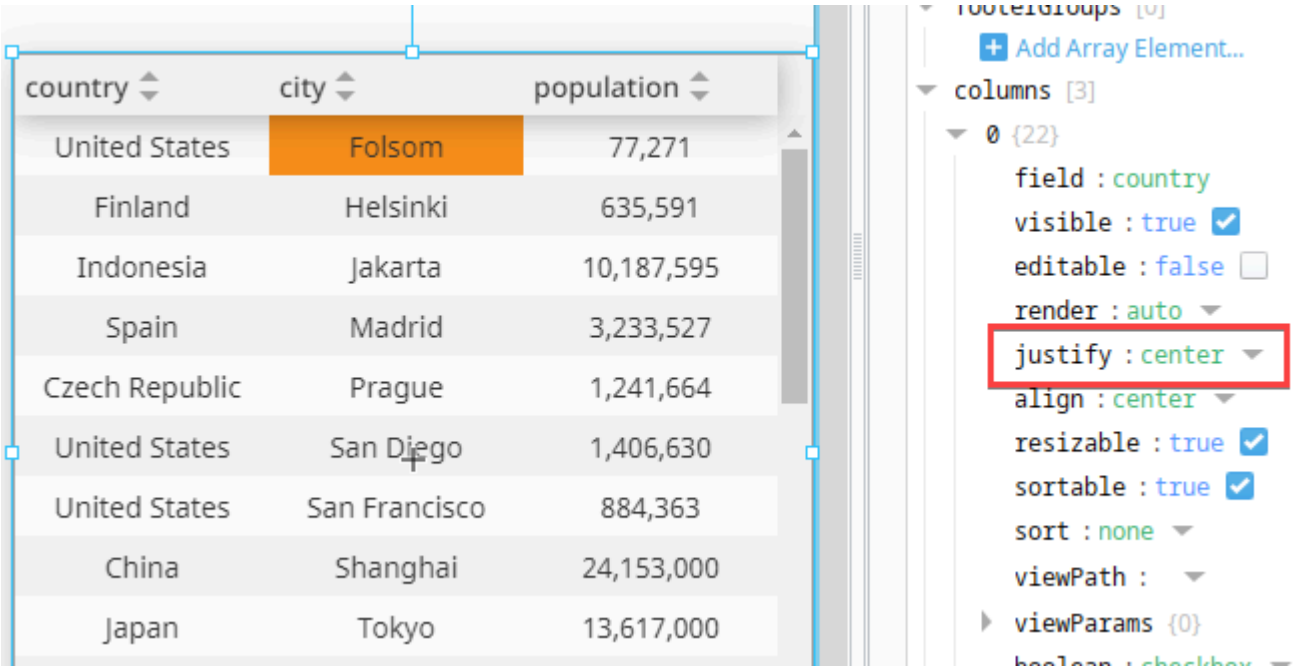
1. Create a new table component, or select an existing one.

2. Add an element for each column you wish to display on the tag to the props.columns property by either clicking the **Add Array Element** button or the + icon right of the **columns** property. Repeat this process until you see all columns on the table again.

3. Map each props.column element to a column in the underlaying data (props.data). For each **props.columns[#].field** property, set the field property to the name of a column/series in our props.data property in the desired column order:

    - props.columns.[**0**].field to **country**
    - props.columns.[**1**].field to **city**
    - props.columns.[**2**].field to **population**

4. Change the alignment on each column by changing the **props.columns.[#].justify** property. In our case, we set the justify property on all columns elements to **center** to center the text.



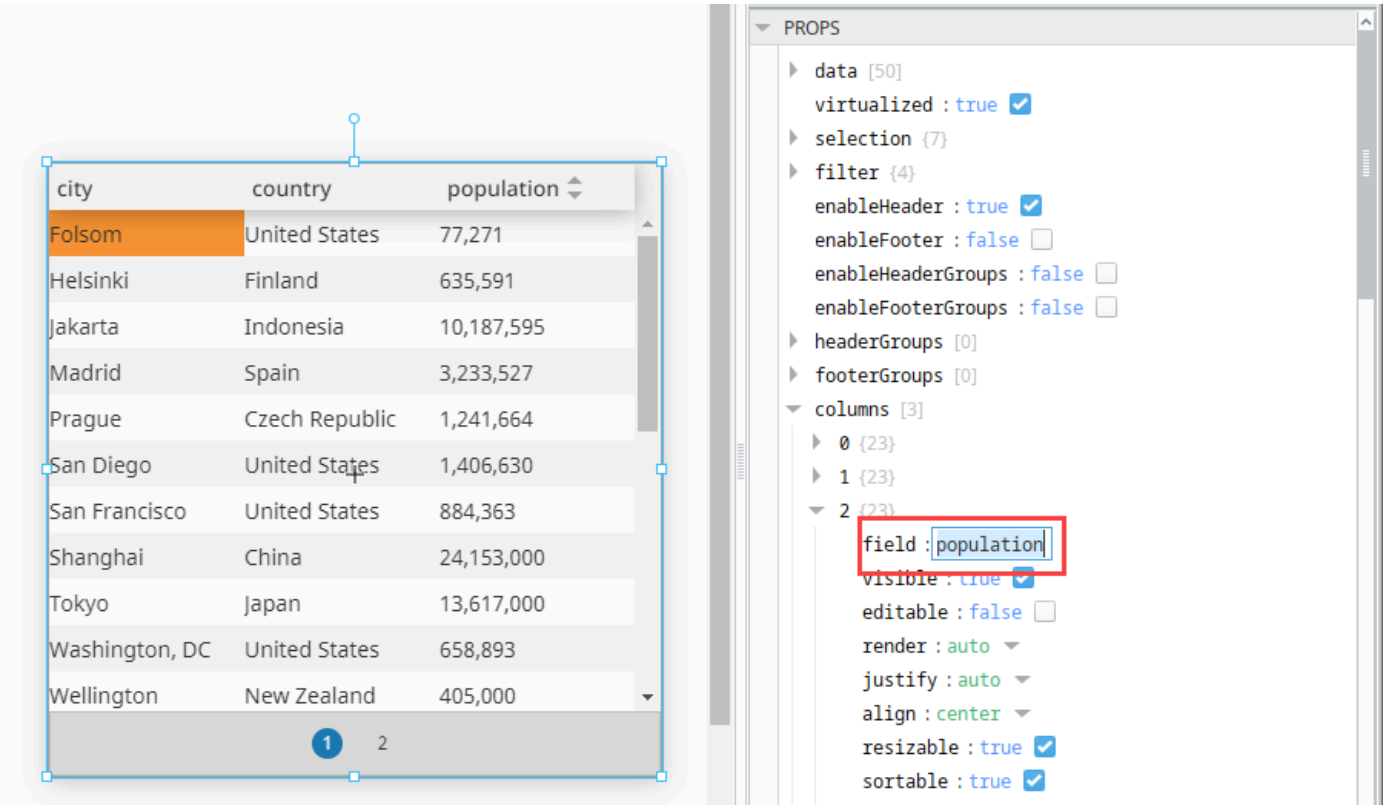
Changing Applied Formatting

In a Perspective Table, you can adjust your column format to display data as desired. In this example, instead of using the default **number [1,000.12]** format to display population counts, we will select **abbreviation [1.2k]** using the **numberFormat** dropdown.

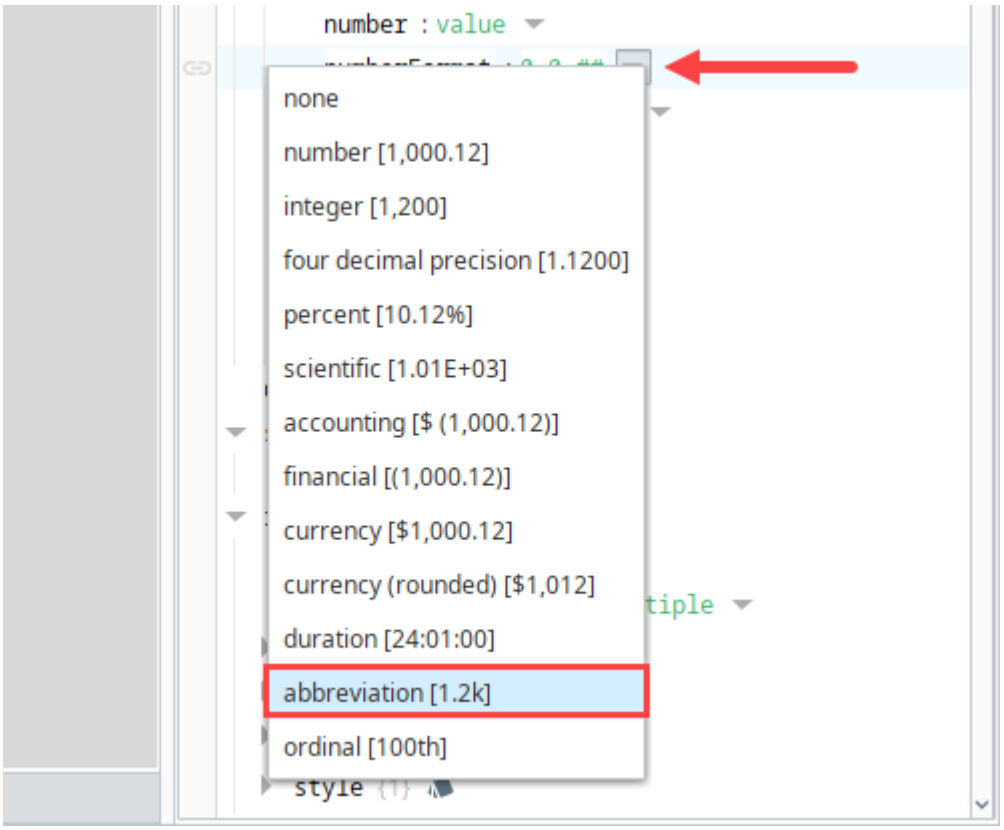
1. Drag a Perspective Table component onto your view. We'll use the default population information.

2. Add three array elements to the columns property of your table.

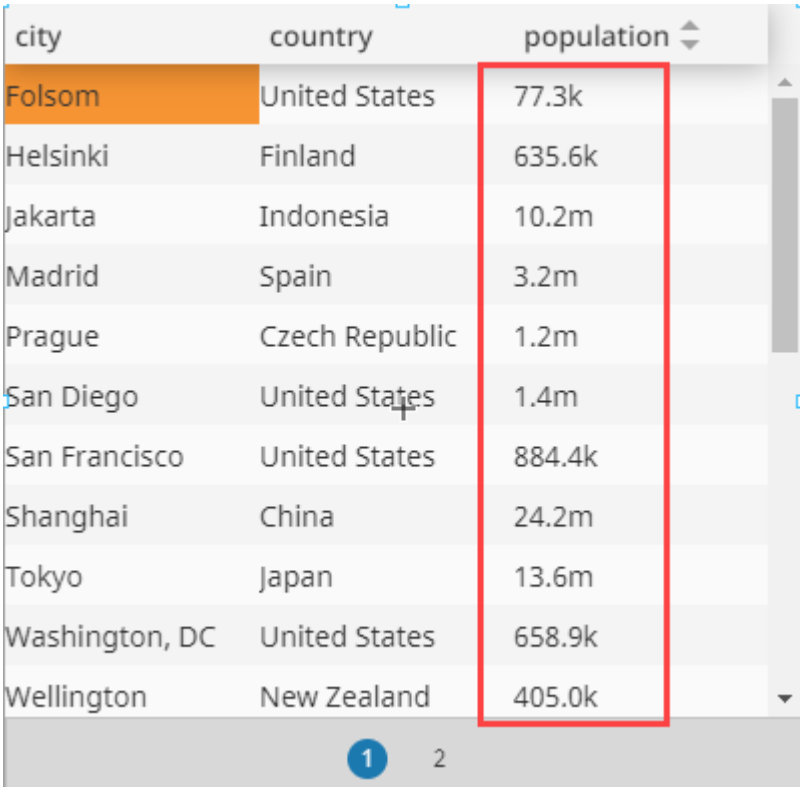
3. Set the **field** property value for props.columns.[**2**] to **population** to associate the column data in the table to the column configuration.



4. Select the **numberFormat** dropdown to view all format options.



5. Select **abbreviation [1.2k]** to adjust the displayed data in the table's population column.



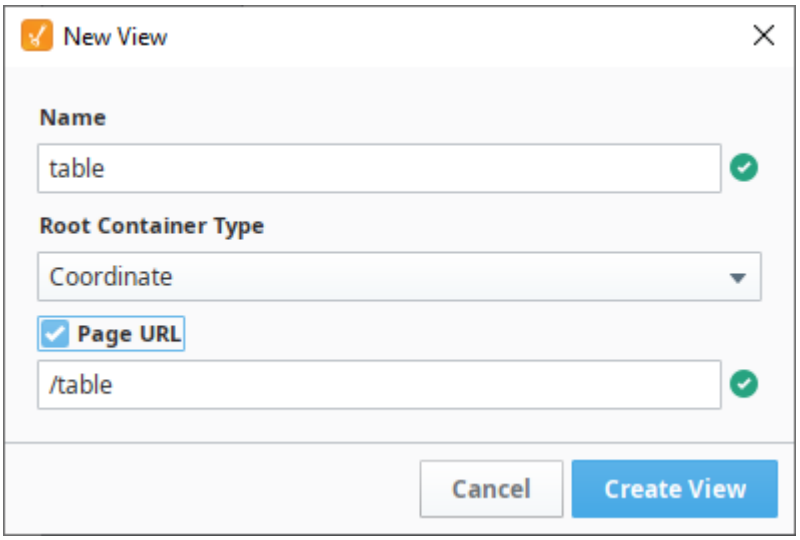
city	country	population
Folsom	United States	77.3k
Helsinki	Finland	635.6k
Jakarta	Indonesia	10.2m
Madrid	Spain	3.2m
Prague	Czech Republic	1.2m
San Diego	United States	1.4m
San Francisco	United States	884.4k
Shanghai	China	24.2m
Tokyo	Japan	13.6m
Washington, DC	United States	658.9k
Wellington	New Zealand	405.0k

1 2

Replacing a Value in a Cell with a Progress Bar

Column configurations can be customized to display a progress bar on the table to show a column's value. We will use the default population information initially configured in the Table component to demonstrate this option.

1. In the Project Browser, right click on Views to create a view. In this example, the view will be named **table**. Set it to have a **Coordinate** root container type.



2. Drag a Table component onto your table view.

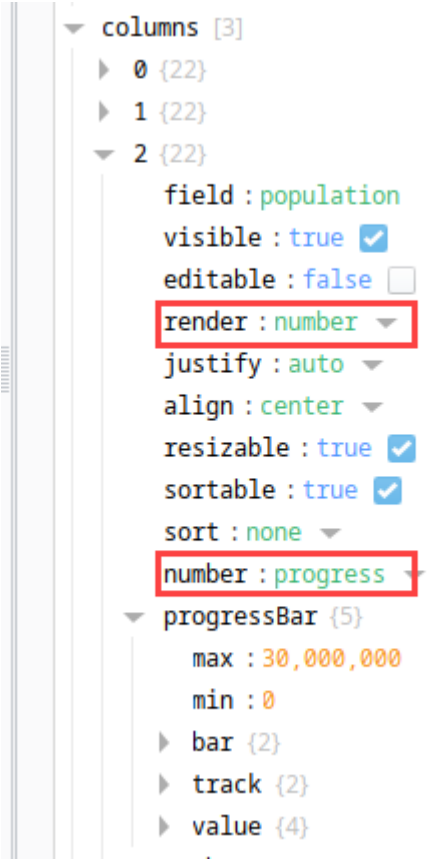
3. Add three array elements to the columns property of your table.

4. **Set the field** property values inside each of the three columns array elements to match each of the column names in your table to associate a column in the table and the custom column configurations.

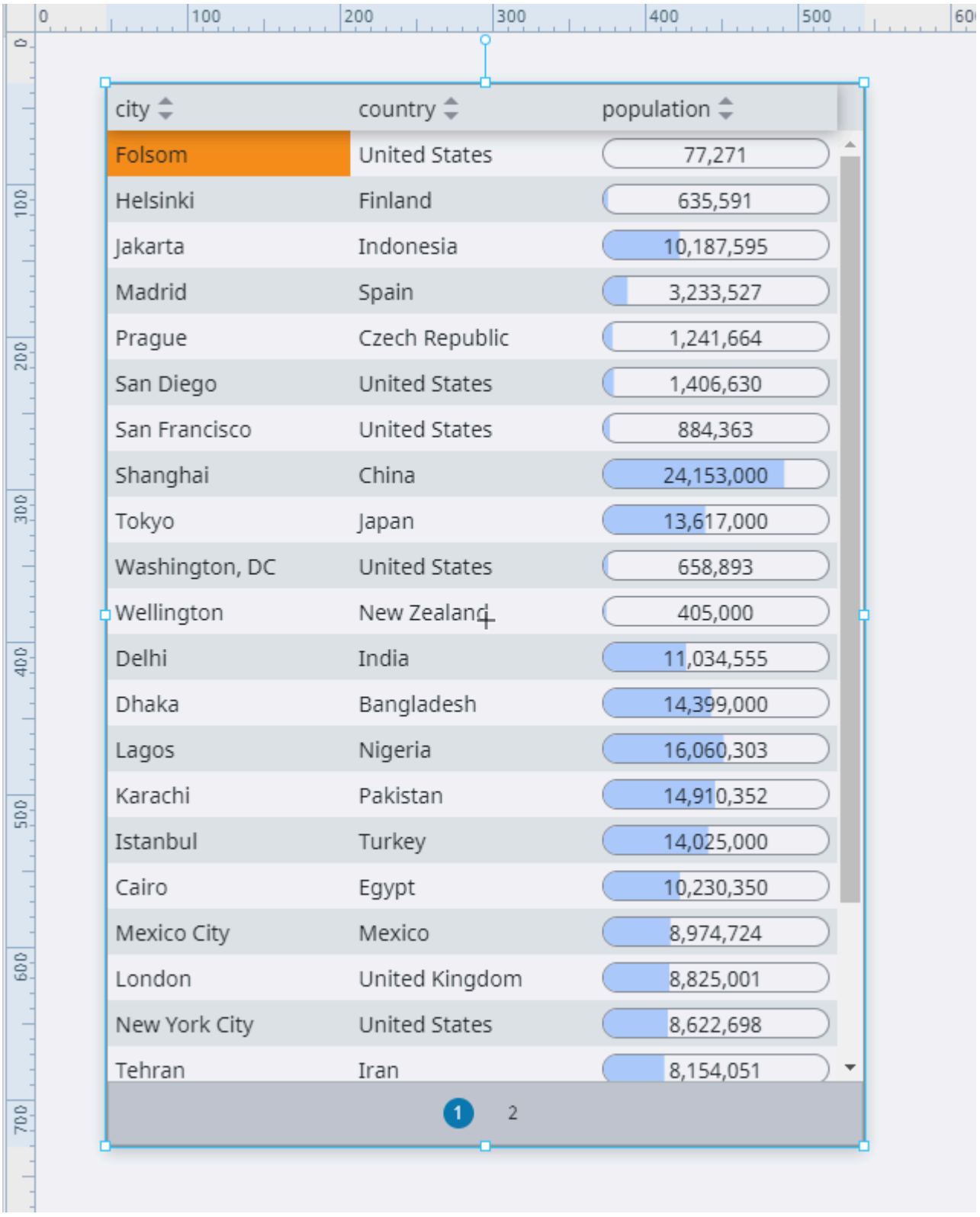
    - props.columns.[**0**].field to **country**
    - props.columns.[**1**].field to **city**
    - props.columns.[**2**].field to **population**

5. Access the props.columns.[**2**] and set the **render** property to **number**.

6. Then, set the **number** property to **progress**.



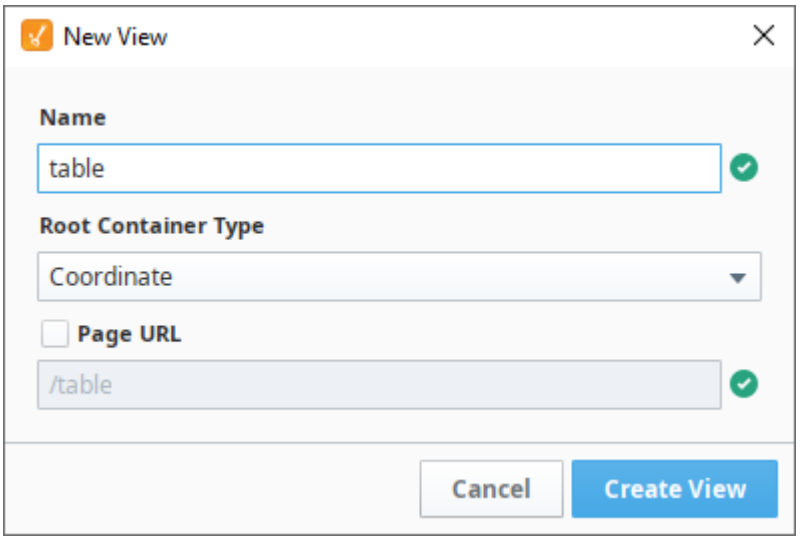
7. Set the **progressBar.max** value to 30,000,000 to account for cities with a large population.



Embedding a View in a Table Cell

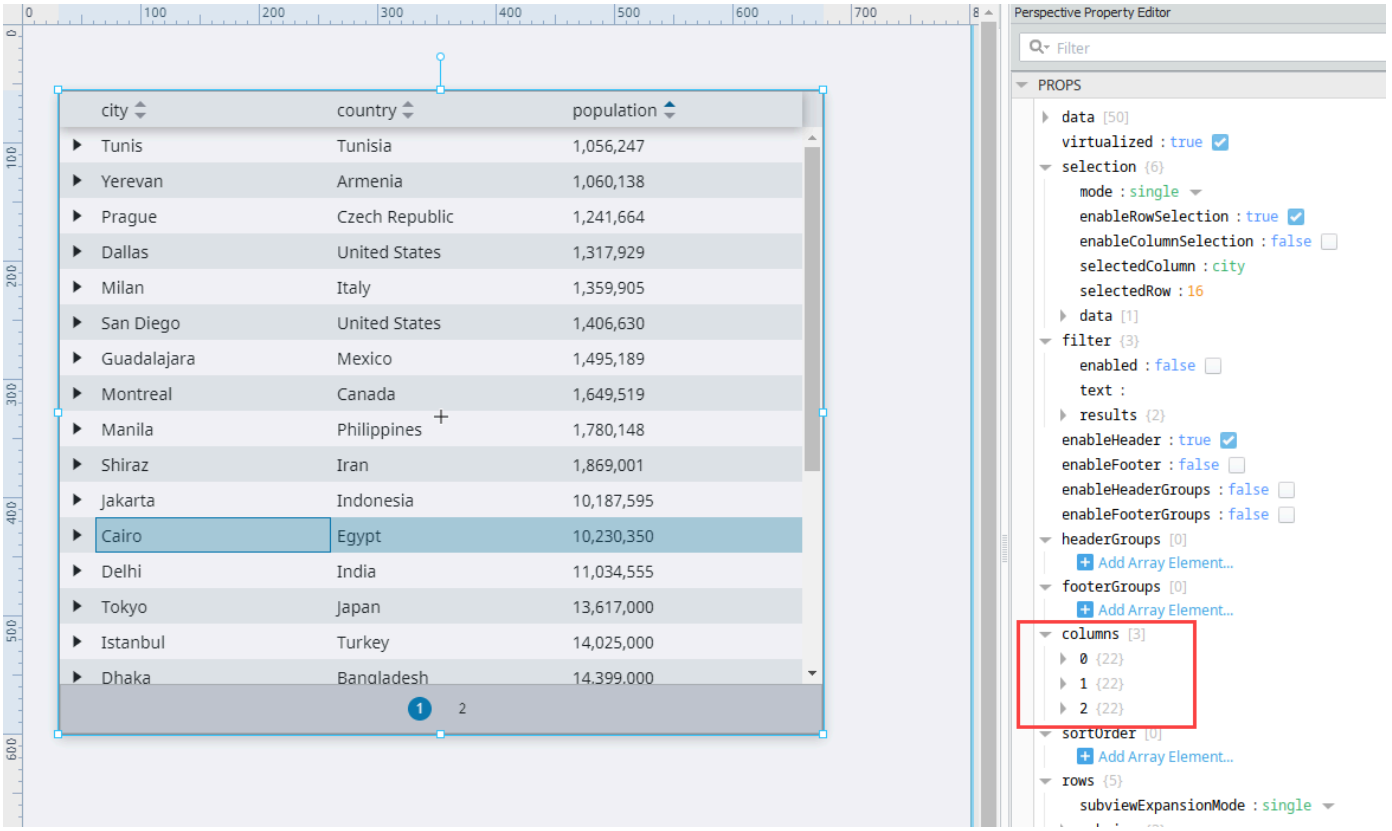
In a Perspective Table, it is possible to embed a view inside a table cell. In this example, instead of using the table's built-in progress bar, we'll embed a view that contains a custom progress bar using the Progress component. The default information that comes on the factory configured table component will be used.

1. Create the table view first. In the Project Browser, right click on Views to create a view. In this example, name the view **table**. Set it to a **Coordinate** root container type.



2. Drag a **Perspective Table** onto the table view.

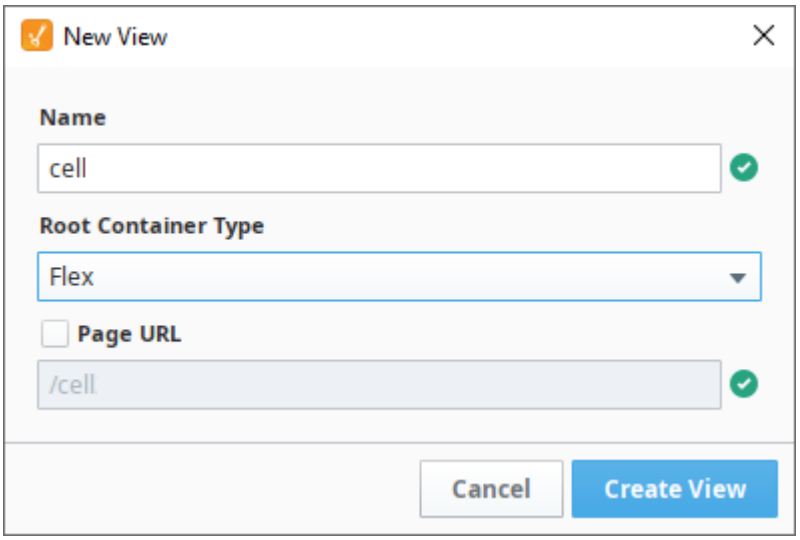
3. Add three array elements to the columns property of the table like as shown in Property Editor as shown in the image below.



4. There is a **field** property inside each of the three column array elements. Set the field values to match each of the column names in your table

    - props.columns.[**0**].field to **country**
    - props.columns.[**1**].field to **city**
    - props.columns.[**2**].field to **population**

5. Now, create the cell view. Right click on Views to create a view. In this example, the view will be named cell. Set it to a Flex Root Container Type.



6. Drag and drop a **Progress** component onto the cell view.

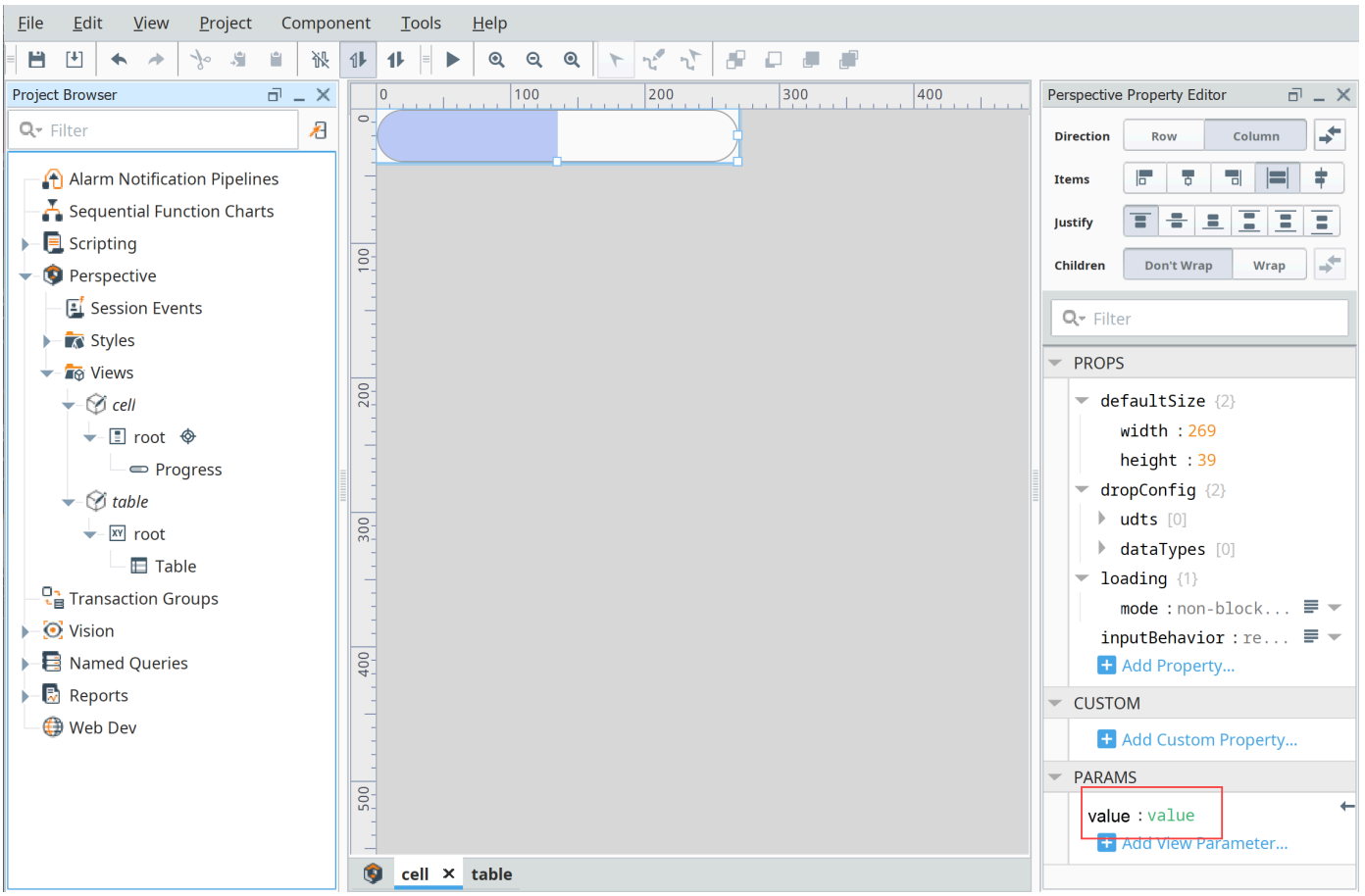
7. Select the Progress Bar component and set the **position.grow** property to **1** so that the bar takes up as much space in the container as possible.

8. In the Project Browser, select the **cell** view, and resize the view so it is closer in size to the population column in the table on the **table** view.

9. Click **Add View Parameter** while your view is selected in the Project Browser.

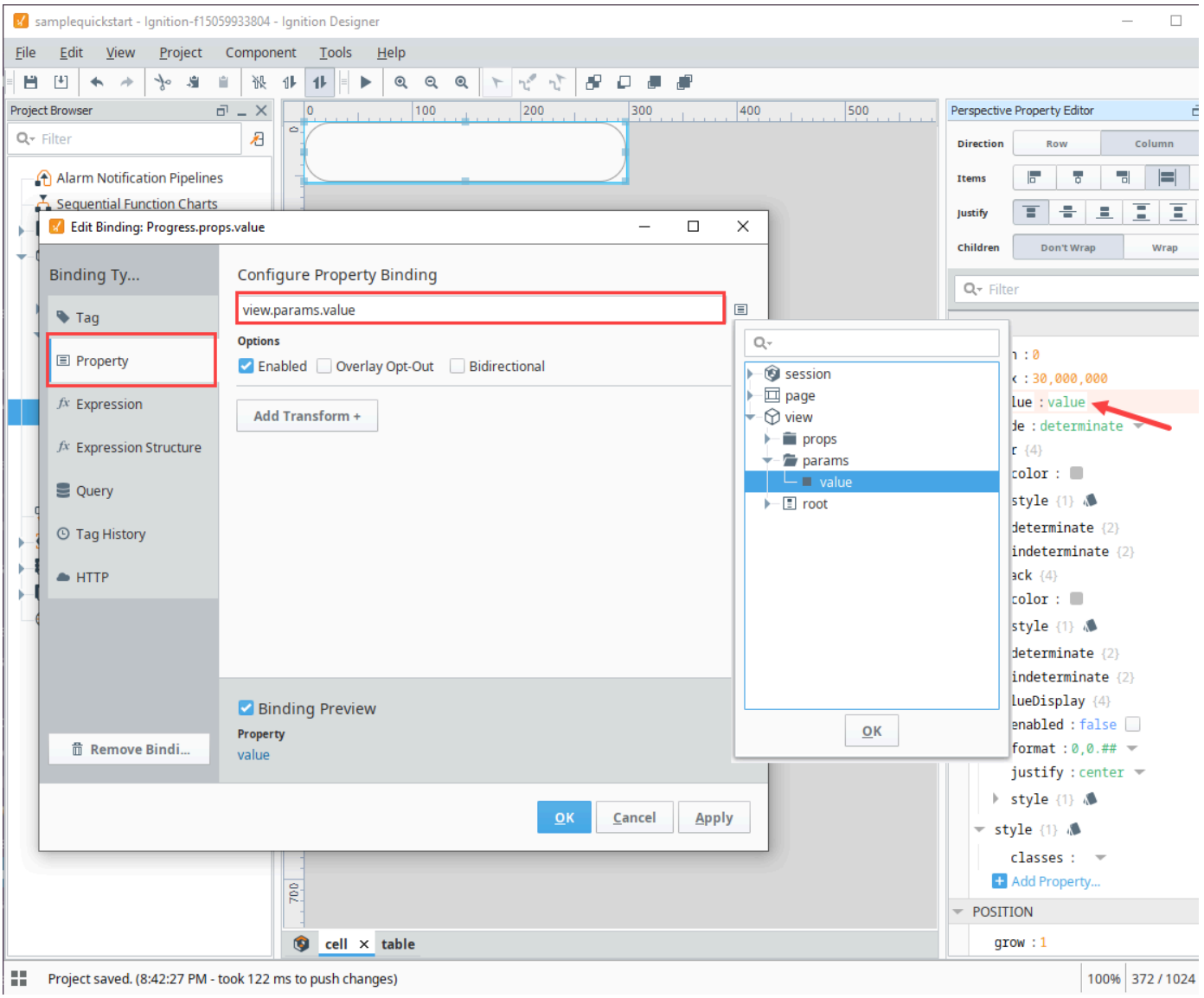
10. Select **Value** from the listed type options.

11. Enter **value** in place of **key**.



12. Set the **max** property value to 30,000,000 on the Progress component to account for large values.

13. Bind the Progress's **value** property to the view parameter created in Step 10 as shown below.

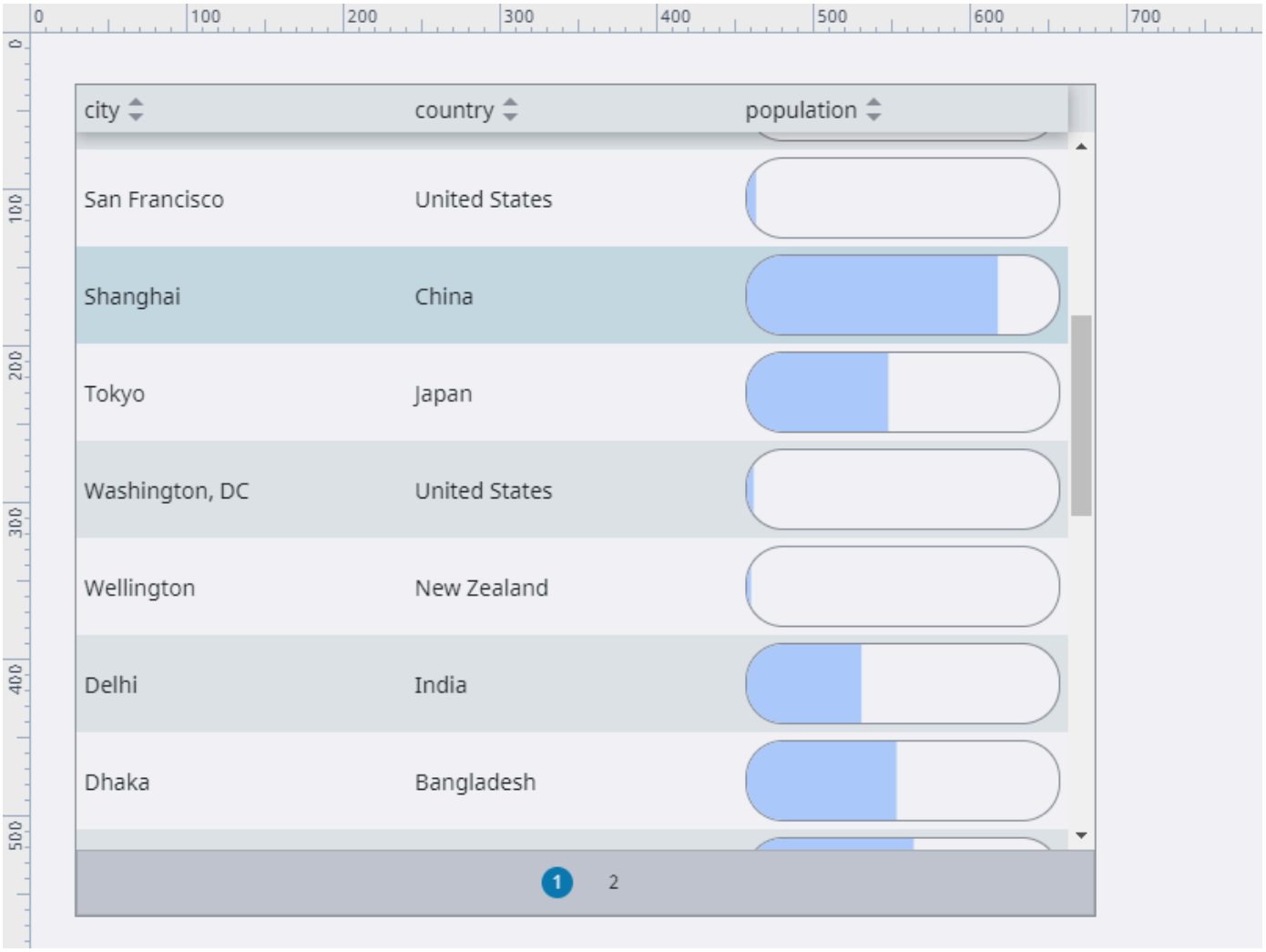


14. Go to the column array element with the **field** value of **population** and set the **render** value to **view**.

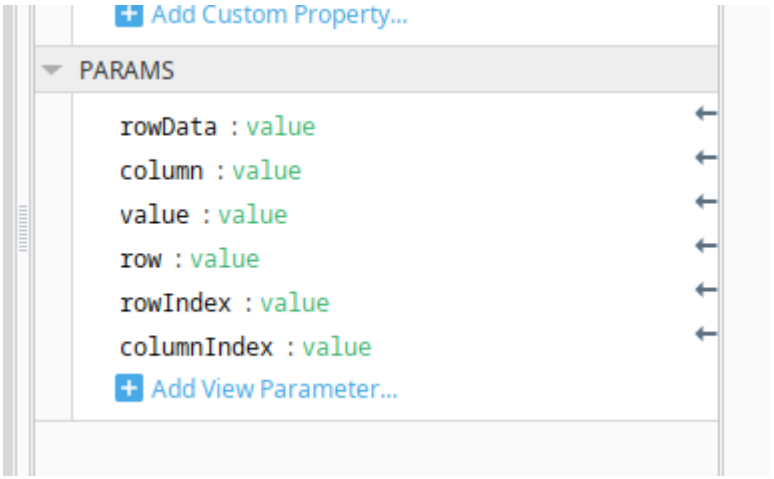
15. Set the **viewPath** to **cell** to embed **cell view** into the population cell values.



16. After the population column is pointed to the cell view, the population number from the table cell will be passed to the cell view. Since the cell view's Progress Bar has its value property bound to the cell view's input parameter, the population value will then be displayed on the table by the Progress Bar in the cell view. If you wanted to resize the progress bar, simply change the **height** and **width** properties under the **defaultSize** property on the **cell** view.

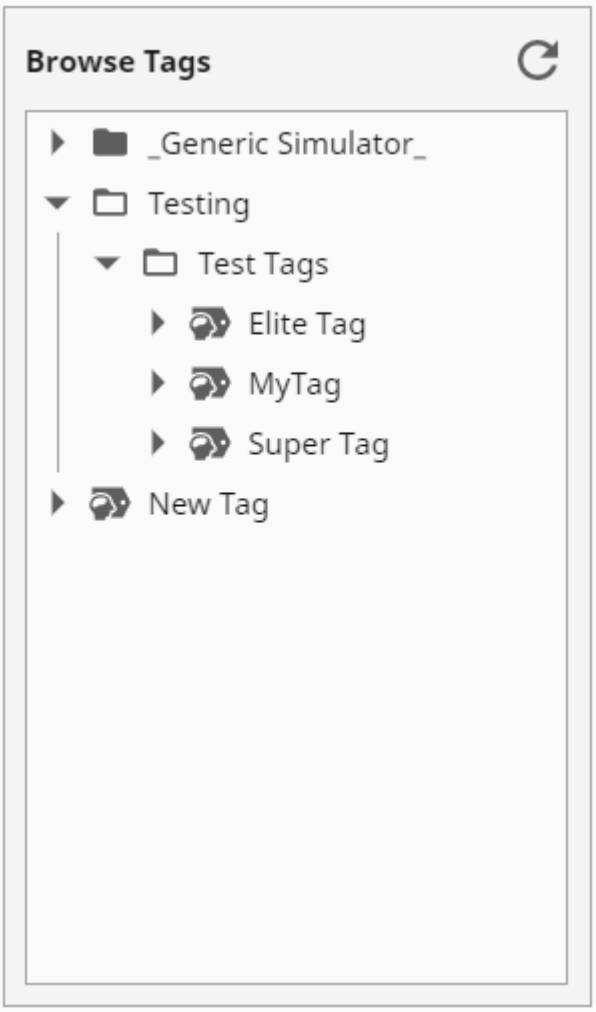


Once a column is configured to render each cell as a view, the table component will automatically pass each view a number of contextual view parameters. Components on the rendered view can access these property values by first configuring view params that match the passed keys.



Property Key	Description
`column`	A string representing the column name of the column, as defined by `props.columns.[columnNumber].field`.
`columnIndex`	A number representing the index of the current column. Similar to columnNumber in `props.columns.[columnNumber]`.
`row`	A number representing the index of the row. Note that this property does not adjust for hidden rows, or rows that are otherwise not displayed.
`rowData`	A JSON object representing table data for the current row. This is equivalent to accessing props.data[row] on the table.
`rowIndex`	A number representing the displayed index of the row. This property differs from **row** in that **rowIndex** does not count rows that aren't displayed. For example, assuming a table is only showing 25 rows per page, then the first row on the second page of results would display a **rowIndex** of 0, while **row** would provide a value of 25.
`value`	Represents the value of the cell being rendered as a view.

Perspective - Tag Browse Tree



Component Palette Icon:

Tag Browse Tree

Description

The Tag Browse Tree component displays a tree hierarchy based on an array of objects. Icons can be chosen for the nodes of the tree, and different icons can be used when an node is expanded or collapsed.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are

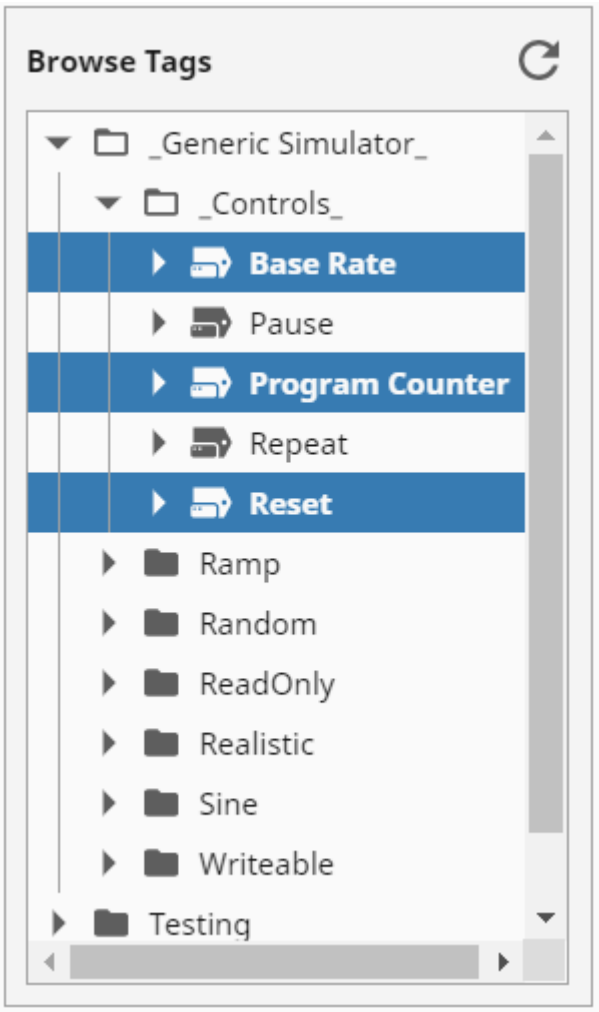
described on the **Perspective Component Properties** page.

Name	Description	Property Type
root	Configuration for the path from which the displaying folder/tag structure will start. • **path**: String value representing the "starting path" from which the tag structure will begin displaying. Value is string.	object
filter	Tree filtering configuration. • **enabled**: Enables the visibility of the filter. Value is boolean. • **text**: The filter text. Value is string.	object
selection	Configuration for the selected tag. • **mode**: Mode used when selecting tags. Choose between single to limit selection to only one tag, or multiple to allow selection of multiple tags at once. Value is string. • **values**: List of the selected tag paths in the order in which selection occurred. Value is an array.	object
display	Display settings for the component. • **refreshIcon**: Display settings for the refresh icon. ◦ **visible**: Visibility setting for the refresh icon.Value is boolean. ◦ **path**: Path to the icon used to represent the "refresh" action. Value is string. ◦ **style**: Sets a style for the refresh icon. Full menu of style options is available. You can also specify a style class.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Scripting

See the **Perspective - Tag Browse Tree Scripting** page for the full list of scripting functions available for this component.

Example



Property	Value
selection.mode	multiple
selection.values.0	_Generic Simulator_/_Controls_/Base Rate
selection.values.1	_Generic Simulator_/_Controls_/Program Counter
selection.values.2	_Generic Simulator_/_Controls_/Reset

Perspective - Tag Browse Tree Scripting

This page details the various scripting, component, and extension functions available for Perspective's **Tag Browse Tree** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onNodeClick

Fires whenever a node is clicked.

Object Path	Type	Description
event.name	String	The name of the node that was clicked.
event.path	String	The Tag path of the node that was clicked.

onNodeDoubleClick

Fires whenever a node is double-clicked.

Object Path	Type	Description
event.name	String	The name of the node that was clicked.
event.path	String	The Tag path of the node that was clicked.

onNodeContextMenu

Fires whenever a node is right-clicked.

Object Path	Type	Description
event.name	String	The name of the node that was clicked.
event.path	String	The Tag path of the node that was clicked.

Component Functions

This component does not have component functions associated with it.

Extension Functions

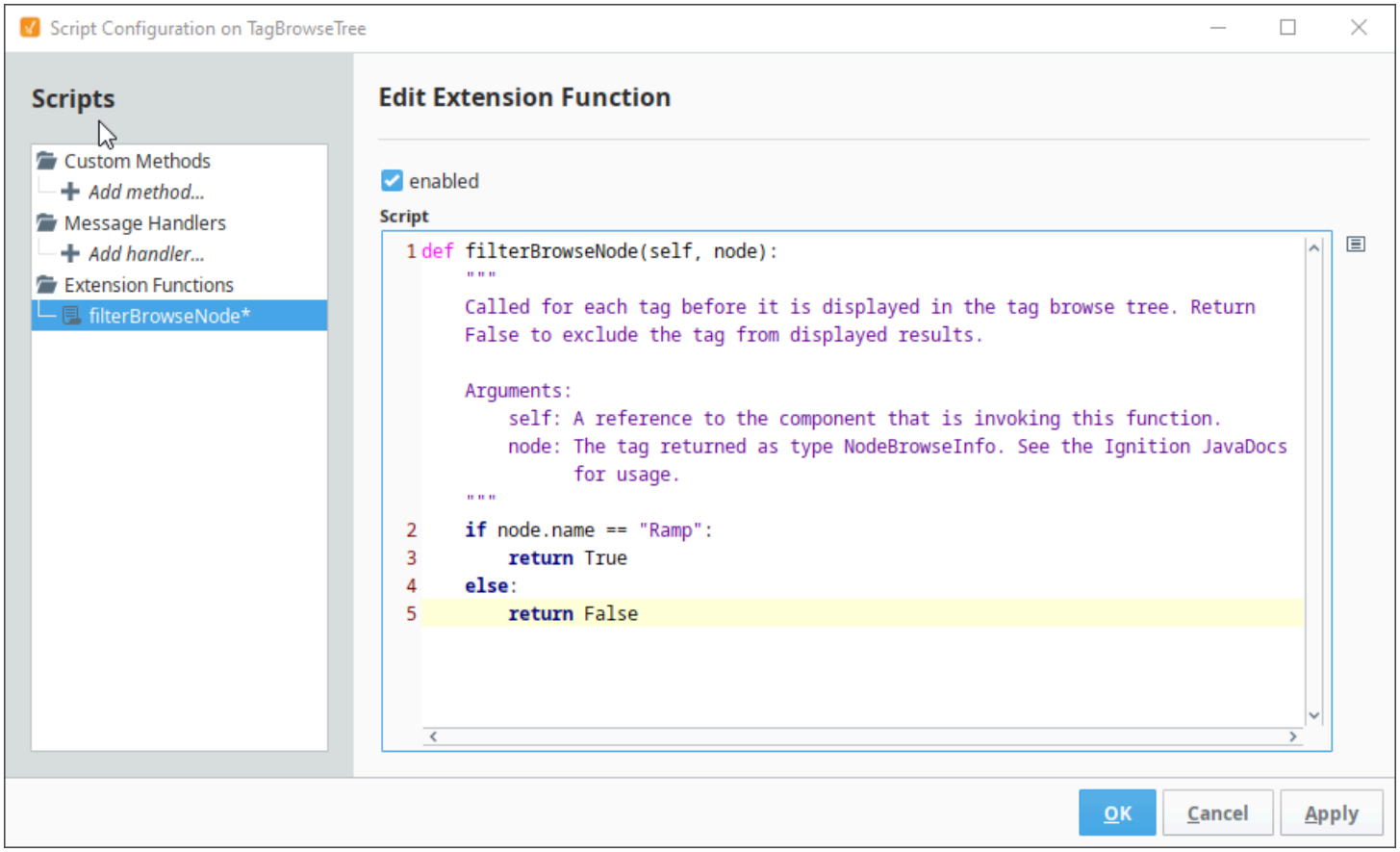
filterBrowseNode

- Description
  - Called for each Tag before it is displayed in the Tag Browse Tree. Provides an opportunity to create a complex filter for the Tags in the Tag Browse Tree. filerBrowseNode is best used alongside the Tag Browse Tree's root.path property to specify where the filter should begin filtering. Return False to exclude the Tag from displayed results.
- Parameters
  - ComponentModelScriptWrapper.SafetyWrapper **self** - A reference to the component that is invoking this function.
  - NodeBrowseInfo **node** - The Tag returned as type NodeBrowseInfo. See the Ignition JavaDocs for usage.
- Return
  - Boolean - The function must return either a True or False.

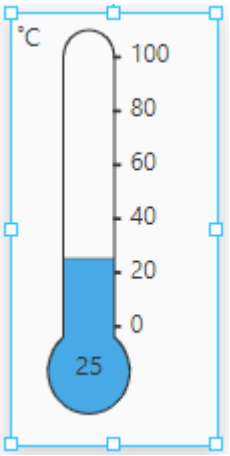
Example

filterBrowseNode Example

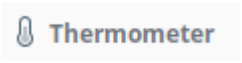
```python
# This example will filter out any nodes (both Tags and folders included) that do not match the string Ramp.
if node.name == "Ramp":
    return True
else:
    return False
```



Perspective - Thermometer



Component Palette Icon:



Description

The Thermometer component displays a temperature value depicted as a level in a mercury thermometer. Temperature intervals can be defined with their own colors so that the mercury color changes based on the temperature range. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

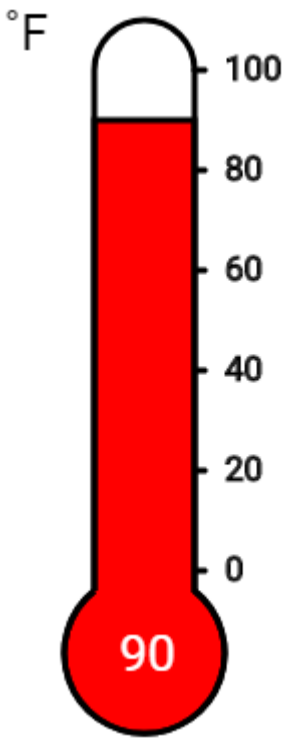
Name	Description	Property Type
thermometerColor	Color of the outline of the thermometer. Default is black.	color
mercuryColor	Color of the mercury. See **Color Selector**.	color
axisLabelColor	The color of the thermometer's y-axis label. Default is black. See **Color Selector**.	color
strokeWidth	Width of the lines used to draw the thermometer in pixels.	value: numeric
highBound	The high boundary value for the whole thermometer.	value: numeric
lowBound	The lower boundary value for the whole thermometer.	value: numeric
value	The value to display in the thermometer. The mercury level and value label will change to reflect this.	value: numeric
unit	A string to describe the units for the current value label. Options are "F" for Fahrenheit or "C" for Celsius.	value: string dropdown
valueFontcolor	The color of the current value. See **Color Selector**.	color
valueFont	The font to use for the current value label. • **fontSize**: Size of the font for the current value. Value is numeric.	object
intervals	Defines the upper and lower temperature range for each interval. • **color**: Color of the mercury in the thermometer. See **Color Selector**. • **high**: High bound value for the interval. Value is numeric. • **low**: Low bound value for the interval. Value is numeric.	object

Name	Description	Property Type
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

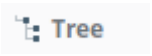


Property	Value
props.unit	F
props.mercuryColor	#8AFF8A
props.intervals.1.high	45
props.intervals.0.color	#0000FF
props.intervals.1.color	#CCCCFF
props.intervals.2.color	#FF0000
props.intervals.2.low	85
props.value	90

Perspective - Tree



Component Palette Icon:



Description

The Tree component displays a tree hierarchy based on an array of objects. Icons can be chosen for the nodes of the tree, and different icons can be used when an node is expanded or collapsed.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
items	An array of objects, each of which represents a node on the tree. • **label**: Label text for the list item. Value is string. • **expanded**: Whether or not the tree appears with all levels expanded. Value is boolean. • **data**: String data for list item. Value is string. • **items**: An array of objects, each of which represents a child node on the tree. Value is an array.	array
interactable	If set to false, the tree is displayed but the user can't interact with it in the runtime. Default is true.	value: boolean
selection	Holds the item index path of the current selection.	value: string
selectionData	Array of objects containing the data and index path for all currently selected nodes. • **itemPath**: Index path. Value is numeric. • **value**: The value of the 'data' property for the selected node. Value is string.	array
appearance	Settings for the appearance of the tree. Click here to see the **appearance** properties.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

appearance Properties

Name	Description	Property Type
textOverflow	Setting indicating whether overflowing text should cause the entire tree to scroll horizontally or whether the text should be truncated with an ellipsis. Default is scroll.	value: string dropdown
expandIcons	Settings for the expand icons . Options as follows: • **collapsed**: Icon appearance when path is collapsed. ◦ **path**: Path to the icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. ◦ **Color**: Fill color to apply to the icon. ◦ **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **expanded**: Icon appearance when path is expanded. ◦ **path**: Path to the icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. ◦ **Color**: Fill color to apply to the icon. ◦ **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **empty**: Icon appearance when path is empty. ◦ **path**: Path to the icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. ◦ **Color**: Fill color to apply to the icon.	object

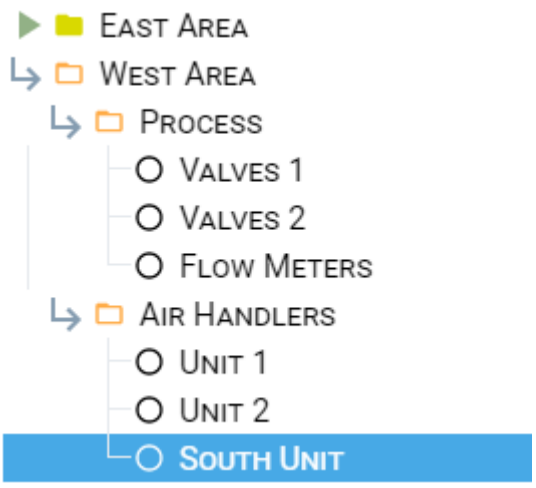
Name	Description	Property Type
	◦ **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
defaultNodeIcons	Settings for the node icons. Options as follows: • **collapsed**: Icon appearance when path is collapsed. ◦ **path**: Path to the icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. ◦ **Color**: Fill color to apply to the icon. ◦ **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **expanded**: Icon appearance when path is expanded. ◦ **path**: Path to the icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. ◦ **Color**: Fill color to apply to the icon. ◦ **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **empty**: Icon appearance when path is empty. ◦ **path**: Path to the icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. ◦ **Color**: Fill color to apply to the icon. ◦ **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin	object

Name	Description	Property Type
	and padding, border, shape and miscellaneous. You can also specify a **style class**.	
selectedStyle	Sets a style for when nodes are selected. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
unselectedStyle	Sets a style for when nodes are unselected. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
rowHeight	Height, in pixels, of each row/node of the tree. Default is 24.	value: numeric

Scripting

See the **Perspective - Tree Scripting** page for the full list of scripting functions available for this component.

Example



Property	Value
appearance.defaultNodeIcons.collapsed.path	material/Play_arrow
appearance.defaultNodeIcons.collapsed.color	#D9D900
appearance.defaultNodeIcons.expanded.path	material/subdirectory_arrow_right
appearance.defaultNodeIcons.expanded.color	#FFAC47
appearance.defaultNodeIcons.empty.path	material/panorama_fish_eye
appearance.defaultNodeIcons.empty.color	#000000

Perspective - Tree Scripting

This page details the various scripting, component, and extension functions available for Perspective's **Tree** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onItemClicked

Fires whenever a node is clicked.

Object Path	Type	Description
event.data	Any	The value of the contextual 'data' object on the clicked node.
event.itemPath	List	A list containing the item indexes leading to the item that was clicked.
event.label	String	The displayed text on the clicked item.

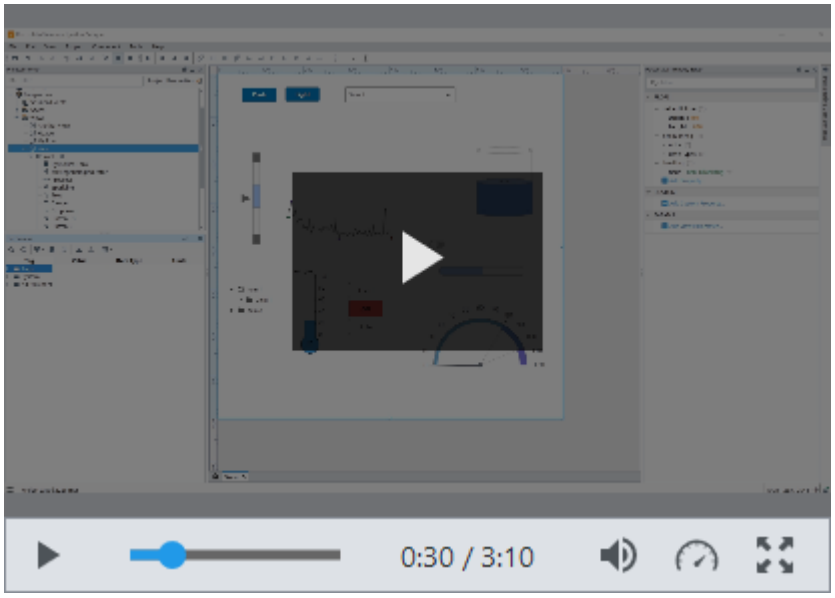
Component Functions

This component does not have component functions associated with it.

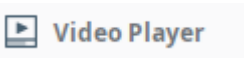
Extension Functions

This component does not have extension functions associated with it.

Perspective - Video Player



Component Palette Icon:



Description

The Video Player component enables you to embed video or a live feed in Perspective views. In a view, the component displays either a live feed from an IP camera or a web hosted video file that is accessible from your gateway. The component is wrapped in a skin that gives you control over the style of the video controls and a uniform experience across browsers. Video controls can also be hidden (available on hover) to allow for a simple, clean video display.

The component requires a URL to a video or live feed. This also includes files placed on a **WebDev** mounted folder or file resource, which can be used to serve video files.

Designer Playback

The Designer contains an instance of JxBrowser to display your views as you build them. There are a **few codecs that JxBrowser does not support**. Because of this, you may find that some videos do not play or display correctly while in the Designer. This is only a limitation of the codecs available to the Designer. The video will work as expected in a client session assuming it supports the required codec.

> **NOTE**
>
> This component plays embedded media files, which is not supported by the Safari 14 web browser. As a result, Sessions running in Safari 14 will not be able to utilize video playback on this component.

Mobile Platform Restrictions

Due to security restrictions on some mobile platforms (and in certain use cases), there are some special behaviors to be aware of when using this component.

iOS

All iOS devices require user interaction (touch, click, etc) to play the video. For this reason, the `controls.play` parameter will not play or pause the video. That must be done by the user clicking the play button. Because of this restriction, this platform also will only use the native look of the player (as determined by the web browser), rather than the custom look that is provided by the Perspective module.

iOS and Android Tablet:

On these platforms, security restrictions surround the use of the `controls.autoplay` property. Video content can only be automatically played if there is no audio. Because of this restriction, the `controls.mute` property must also be set to true.

Fullscreen Mode (all desktop and mobile platforms):

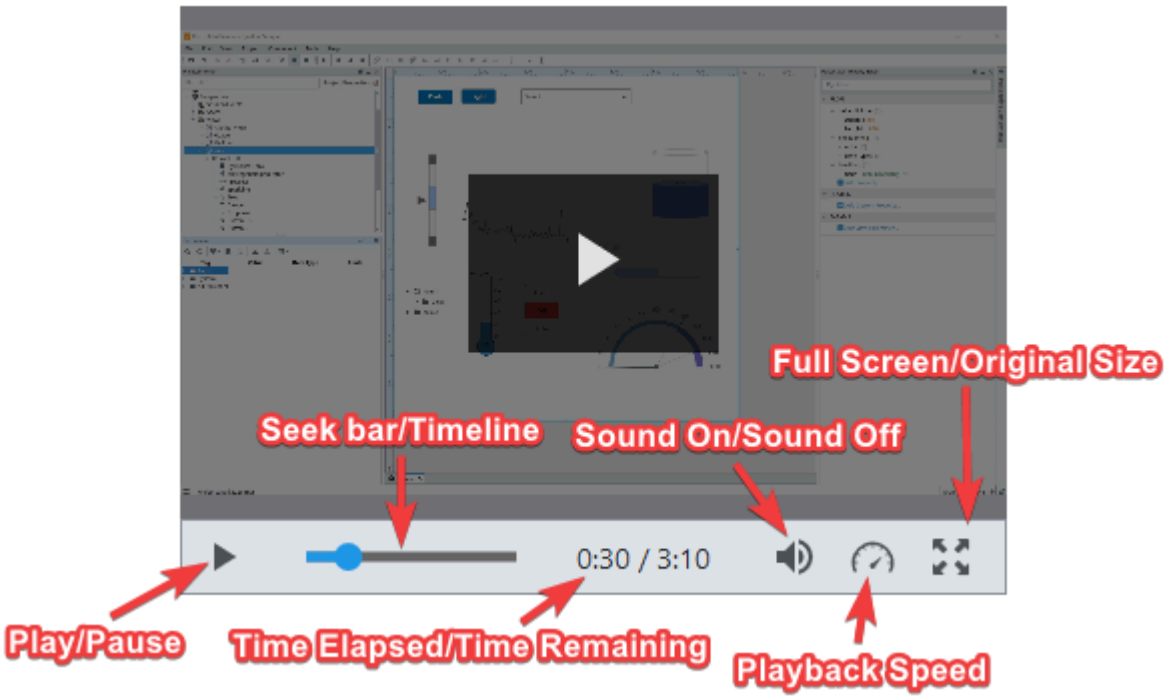
When in fullscreen mode, the native look of the player (determined by the web browser) is used as opposed to the custom look provided by the Perspective module. Because of this, the `controls.play` parameter will not play or pause the video. That must be done via user action (click, touch, etc).

User Interaction

The Video Player component properties have impact on the way a user can interact with it in the runtime.

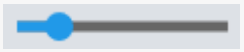
Interaction	Description
Viewing on a Mobile Device with Android	On Android, you'll get the same experience as the desktop display with one exception; when going into fullscreen mode, you'll be presented with the native video control for a cleaner fullscreen experience on that platform.
Viewing on a Mobile Device with iOS	On iOS, you'll get the native video control for standard and fullscreen mode.

User Interface



Controls

The following controls are available to the user in a session.

Icon	Definition	Description
	Play	Starts the video play.
	Pause	Pauses the video play.
	Playback Speed	Sets the speed of the playback. Options are .25, .5, Normal, 1.25, 1.5, 2, 5, and 10 (for example, .5 is half speed, 2 is double speed, etc.).
	Seek bar/Timeline	Interactable slider representation of the time elapsed and time remaining.
	Sound On	Sound is turned on for the video. Clicking on this icon brings up a sliding bar with which you can adjust the volume.
	Sound Off	Sound is turned off for the video.
	Time elapsed/Time remaining	Displays the time elapsed in the video and the time remaining.
	Full Screen	Expands the video to full screen.
	Original size	Returns the video to original screen size. You can also press the Esc key to return to original size.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
source	The path to the source of the video or live feed.	value: string
liveFeed	Used to toggle the component to display a live feed. If set to true, the poster, autohideControls, controls, and status properties will be hidden as they pertain only to a video file.	value: boolean
poster	The path to an image that will display as the background image of the video file when the video has not yet loaded. (Hidden if props.liveFeed is set to true.)	value: string
autohideControls	Used to toggle the visible state of the control bar when displaying a video file. If set to `true`, the control bar will be displayed only when the mouse is hovered over the video. (Hidden if props.liveFeed is set to true.)	value: boolean
controls	Properties that are used to provide settings and interaction points with a video file. (Hidden if props.liveFeed is set to true.). • **autoplay**: If set to true, the video will begin playing when the client session loads. While autoplay is enabled, the video will be muted while playing initially. Value is boolean. • **loop**: If enabled, the video file will play over indefinitely once it has completed. Value is boolean. • **mute**: If enabled, the video will show as being in a muted state and will have no volume. Value is boolean. • **play**: If enabled, the video will play. If disabled, the video will be paused. Value is boolean. • **seek**: The time (in seconds) from which the video should start playing. Value is numeric. • **volume**: A number (percentage value) representing the current volume of the video file. Value is numeric. • **playRate**: The speed at which the video will be played (where 1 is normal speed, .5 is half speed, 2 is double	object

Name	Description	Property Type
	speed, etc.). Value is numeric.	
status	This property holds several sub-properties that are used to provide status updates while the video file goes through the playback process. These sub-properties should not be set as they are constantly overwritten during the playback process. (Hidden if props.liveFeed is set to true.) • **loadedData**: True when the current playback position of the media has finished loading; often the first frame. Value is boolean. • **playing**: True when playback is ready to start after having been paused or delayed due to lack of data. Value is boolean. • **paused**: True when playback has been paused. Value is boolean. • **progress**: A number representing the time (in seconds) where playback has occurred. Value is numeric. • **rateChanged**: A number representing the current playback rate (1 being normal speed). Value is numeric. • **seeking**: True when a seek operation is in progress. Value is boolean. • **seeked**: A number representing the time (in seconds) where the seek operation was completed. Value is numeric. • **waiting**: True when playback has stopped because of temporary lack of data. Value is boolean. • **ended**: True when playback has stopped because the end of the media was reached. Value is boolean.	object
controlStyle	Sets a style for the controls on this component: the control bar, all controls, error messaging, context menus, and control popups. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. Classes are predefined styles in a project.	object

Name	Description	Property Type
style	Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Perspective - Embedding Palette

The following components function a bit differently, but what they all have in common is that each component can be embedded in multiple views of a project.

The Carousel component allows you to display a selection of rotating views. An Embedded View is an instance of a view that is used as a component within another view. The Flex Repeater component lets you easily create multiple instances of components for display in another view, each having the same look, feel, and functionality of the original components.

Here is a complete list of the embedding components, and a link pointing to a page containing the component's description, properties, and an example of how to configure it.

Accordion

Carousel

Embedded View

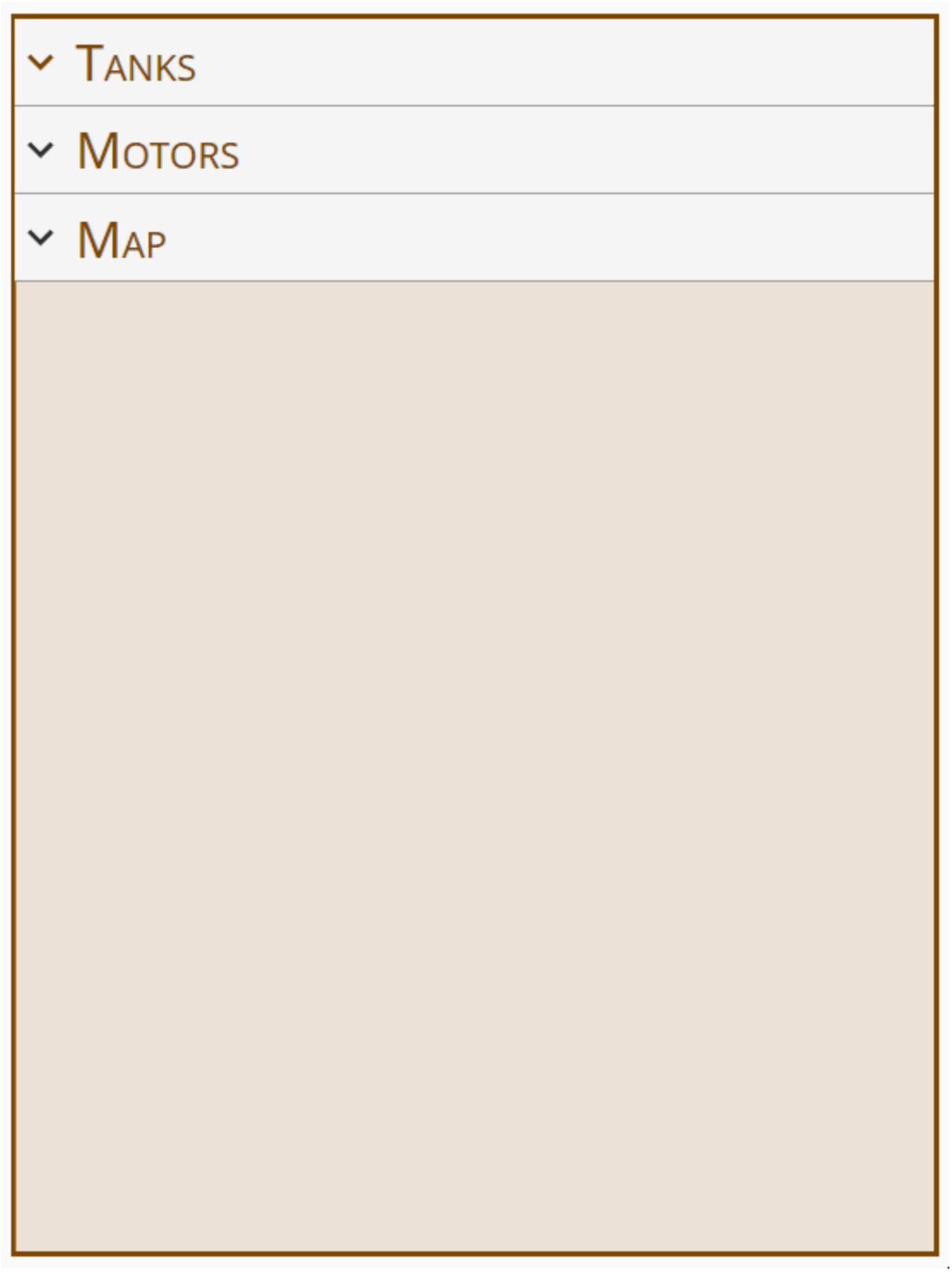
The Embedded View component allows you to include an entire view inside another.

Flex Repeater

The Flex Repeater component lets you easily create multiple instances of views for display in another view.

View Canvas

Perspective - Accordion



Component Palette Icon:



Description

The Accordion component allows the embedding of expandable/collapsible views which can be toggled with a click or a tap of their headers. The headers may contain text or a view.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
expansionMode	Determines how many items can be expanded at a given time. Options are `single` and `multiple`. When using `single` any item that's currently expanded will collapse if another one is clicked. When using `multiple`, items that are expanded will remained open until they are collapsed.	value: string
items	An array of items in the accordion. Each item has a separate header and body configurations. • **expanded**: Determines if the the accordion body expanded. Set to true to expand, false to collapse. Value is boolean. • **header**: An object containing configuration options for the toggle icon. Click to see **header** properties. • **body**: An object containing configuration options for the body. Click to see **body** properties.	array
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
unusedSpaceStyle	Sets a style for the empty area at the bottom of the accordion. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

header Properties

Name	Description	Property Type
toggle	An object containing configuration options for the toggle icon. • **enabled**: Enables the collapse and expand toggle. Value is boolean. • **expandedIcon**: An object containing configuration options for the header icon while the item body is expanded. ◦ **path**: Shorthand path to the icon source, in format: library/iconName. Value is string. ◦ **color**: Color of the icon. May instead 'fill' in the styles prop. Value is string. ◦ **style**: Sets a style for the expandedIcon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **collapsedIcon**: An object containing configuration options for the header icon while the item body is collapsed. ◦ **path**: Shorthand path to the icon source, in format: library/iconName. Value is string. ◦ **color**: Color of the icon. If deleted, the Shape "fill" property in the adjacent style object will determine the color of the icon. Value is string. ◦ **style**: Sets a style for the expandedIcon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
content	An object containing configuration options for the content. • **type**: Whether text or a view will be rendered in this accordion header. Set this property to either 'text' or 'view'. Value is string. • **text**: Text to display for this accordion header. Value is string.	object

Name	Description	Property Type
	• **useDefaultViewWidth**: Use of view's default width instead of adjusting based on the content's width. Value is boolean. • **useDefaultViewHeight**: Use of view's default height instead of adjusting based on the content's height. Value is boolean. • **viewPath**: Path to view to render in this according header. Value is string. • **viewParams**: Params to pass to this view rendered in this accordion header. A dropdown list of parameters is available when the user clicks the **Add Object Member** ➕ icon. This makes it easy to add parameters from the rendered view. • **style**: Sets a style for the content. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
height	The height of the header.	value: string
reverse	Reverses the order of the toggle and header content, (i.e., from left side to the right side).	value: boolean
style	Sets a style for the header. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

body Properties

Name	Description	Property Type
viewPath	Path of the view to display. If a path is present in the viewPath property, an Open View  icon will appear that will navigate directly to the view when clicked.	value: string
viewParams	Parameters for the view. If passing parameters into the embedded view, the names here must match the parameters on that view. A dropdown list of parameters is available when the user clicks the **Add Object Member**  icon. This makes it easy to add parameters from the rendered view.	object
useDefaultViewWidth	Use of view's default width instead of adjusting based on the content's width.	value: boolean
useDefaultViewHeight	Use of view's default height instead of adjusting based on the content's height.	value: boolean
height	The height of the body.	value: string
style	Sets a style for the body. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Scripting

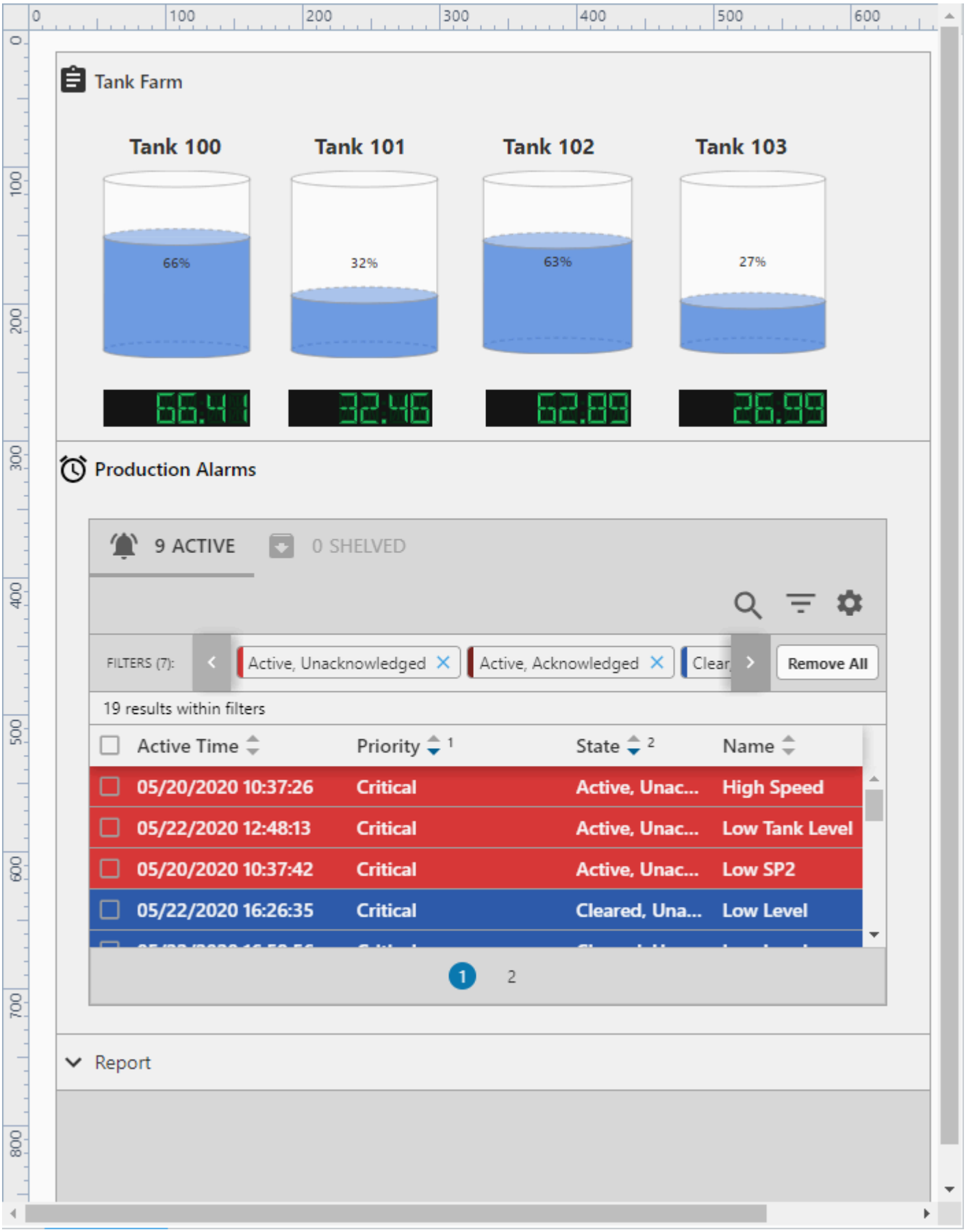
See the **Perspective - Accordion Scripting** page for the full list of scripting functions available for this component.

Example

This Accordion example has three multiple embedded expandable and collapsible views. Each view can be expanded or collapsed by clicking on their headers. The Tank Farm and the Production Alarms views are both expanded while the Report view is collapsed.

The three views used in the accordion example are working views and the component was configured to use these existing views. Configuring the accordion component is just a matter of how you want to present the information on the component, and then to configure its properties.

Preview Mode



Property Settings

Property	Value
expansionMode	multiple
props.items.0.header.toggle.expandedIcon.path	material/assignment
props.items.0.header.toggle.collapsedIcon.path	material/expand_more
props.items.0.header.content.type	text
props.items.0.header.content.text	Tank Farm
props.items.0.body.viewPath	Tank Farm
props.items.0.body.style.marginTop	3px
props.items.0.body.style.margin	10px
props.items.1.header.toggle.expandedIcon.path	material/alarm
props.items.1.header.toggle.collapsedIcon.path	material/expand_more
props.items.1.header.content.type	text
props.items.1.header.content.text	Production Alarms
props.items.1.body.viewPath	Production Alarms
props.items.1.body.style.marginTop	3px
props.items.1.body.style.margin	10px
props.items.2.header.toggle.expandedIcon.path	material/info
props.items.2.header.toggle.collapsedIcon.path	material/expand_more
props.items.2.header.content.type	text
props.items.2.header.content.text	Report
props.items.2.body.viewPath	Report4

Property	Value
props.items.2.body.style.marginTop	3px
props.items.2.body.style.margin	10px

Perspective - Accordion Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Accordion** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onItemExpanded

This event is fired when an item is expanded.

Object Path	Type	Description
event.index	Numeric	The index of the item that was expanded.

onItemCollapsed

This event is fired when an item is expanded.

Object Path	Type	Description
event.index	Numeric	The index of the item that was collapsed.

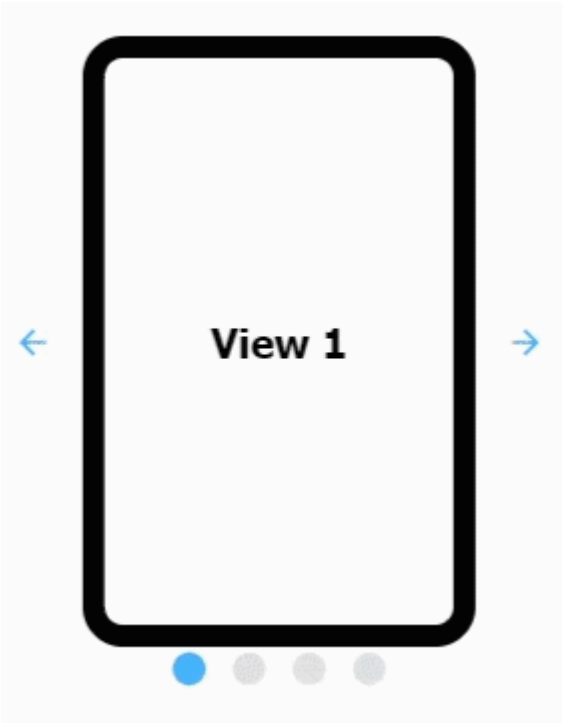
Component Functions

This component does not have component functions associated with it.

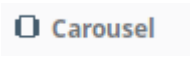
Extension Functions

This component does not have extension functions associated with it.

Perspective - Carousel



Component Palette Icon:



Description

The Carousel component allows you to display a selection of rotating views at a defined rate with a link to the view on a page in your project. The Carousel component can automatically cycle through the views or a user can click through the views on demand, either way, still providing a link to the view on a page.

This version updates how the component handles drag transition ("swiping" across embedded views). Only common rotational angles are supported (90, 180, 270, 360) for drag transitions. If the Carousel's rotation doesn't match one of those angles, then drag transition is disabled.

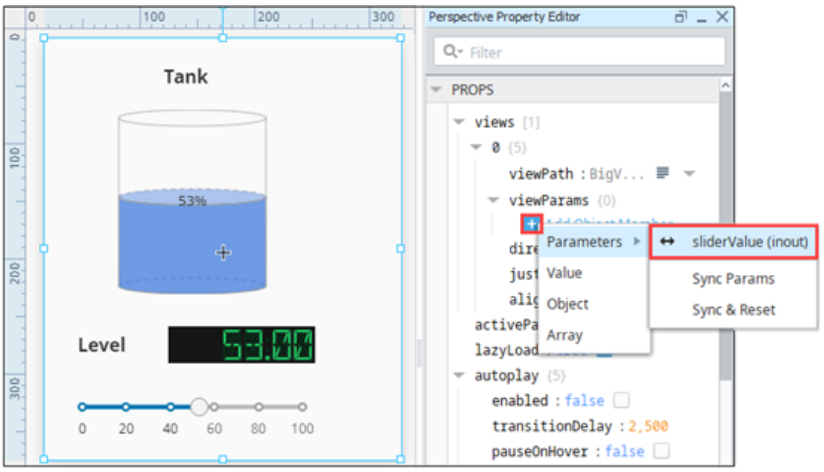
Here are a few best practices when working with the Carousel component.

- Components such as the Video Player and Map are performance intensive components and should not be embedded in the Carousel since they can hurt session performance.
- Avoid embedding views containing carousels in a carousel. This can become confusing for users.
- Avoid embedding views that contain iFrame components. It's easy for content embedded in an iFrame to steal focus from other components. Also, depending on the content in the iFrame, it

may impact performance.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
views	Visible area of a page. Can have multiple views in the carousel component. • **viewPath**: The path of the view to render in this carousel. If a path is present in the viewPath property field, an Open View icon will appear that will navigate directly to the view when clicked. • **viewParams**: Parameters for the view. If passing parameters into the embedded view, the names here must match the parameters on that view. A dropdown list of parameters is available when the user clicks the **Add Object Member** icon. This makes it easy to add parameters from the rendered view.  • **direction**: Direction of the child layout. Options are row or column. Value is string. • **justify**: Adjusts placement of view along the main axis. Options are flex-start, flex-end, or center. Value is string. • **alignItems**: Adjusts placement of view along the cross axis. Options are flex-start, flex-end, or center. Value is string.	value: array
activePane	Active pane being displayed.	value: numeric
lazyLoad	Load views on demand or progressively.	value: boolean

Name	Description	Property Type
autoplay	Settings controlling the rotation of views in the carousel. • **enabled**: If true, the carousel will automatically rotate the views according to the transitionDelay. Value is boolean. • **transitionDelay**: Delay (in ms) at which slides scroll through the carousel when autoplay is true. Value is numeric. • **pauseOnHover**: Pauses autoplay when user hovers the mouse over the view. Value is boolean. • **pauseOnFocus**: Pauses autoplay on focus. Value is boolean. • **pauseOnDotHover**: Pauses autoplay when user hovers the mouse over the dot for the view. Value is boolean.	object
behavior	Behavior and interaction related carousel options. • **transitionSpeed**: The speed (in ms) at which the carousel transitions between slides. Value is numeric. • **fade**: Enables slides to fade in and out of view on transition. Value is boolean. • **mobileSwipeable**: Enables swiping on mobile devices to change slides. Value is boolean. • **desktopDraggable**: Enables scrolling via drag the desktop. Value is boolean. • **swipeThreshold**: The maximum length of a drag in pixels before a slide transition. Threshold will be either the provided value or half the width of the slide pane, whichever is smaller. Minimum value is 50 pixels. Default is 200 pixels. Value is numeric.	object
appearance	Appearance related carousel options. Click here to see the **appearance** properties	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

appearance Properties

Name	Description	Property Type
dots	Carousel dots configuration. • **enabled**: Enables dots at the bottom of the carousel component. Value is boolean. • **iconPath**: Path to the icon that will be used. Value is string. • **styles**: Configure active and inactive styles for the dot icon. ◦ **active**: Sets a style for the dot icon when active. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. ◦ **inactive**: Sets a style for the dot icon when inactive. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
arrows	Carousel arrows configuration. • **enabled**: Enables arrows at the sides of the carousel component. Value is boolean. • **next**: Next arrow icon configuration. ◦ **iconPath**: Path to the icon that will be used for the next arrow, if provided. Value is string. ◦ **fillColor**: Fill color to apply to the icon. Value is string. ◦ **style**: Sets a style for the next arrow. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **previous**: Previous arrow icon configuration. ◦ **iconPath**: Path to the icon that will be used for the next arrow, if provided. Value is string.	object

Name	Description	Property Type
	◦ **fillColor**: Fill color to apply to the icon. Value is string. ◦ **style**: Sets a style for the next arrow. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
useDefaultViewWidth	Enables the use of the view's default width instead of dynamically adjusting based on the available width.	value: boolean
useDefaultViewHeight	Enables the use of the view's default height instead of dynamically adjusting based on the available height.	value: boolean
slidesToShow	Number of views to show on each carousel page.	value: numeric
slidePadding	Applies padding between slides.	value: numeric
reverse	Reverses the slide order. Meaning, the first view rendered on the component will be the last element in the views array property.	value: boolean

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

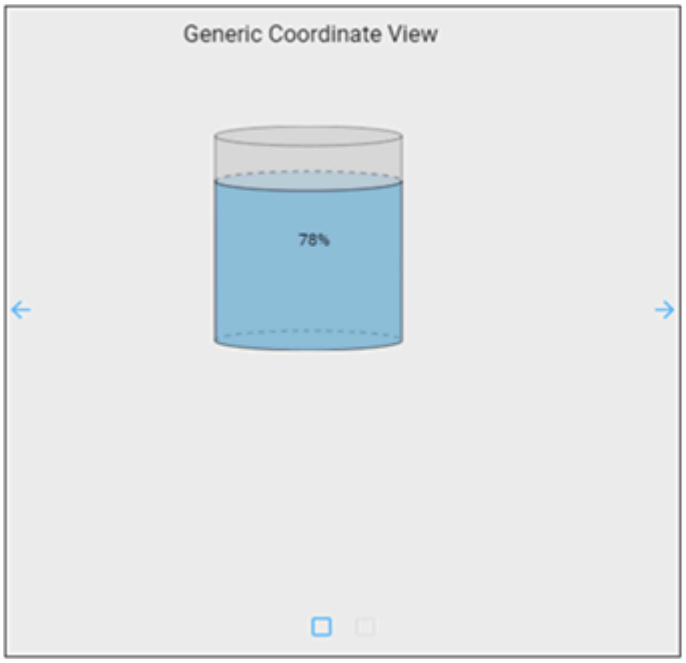
Examples

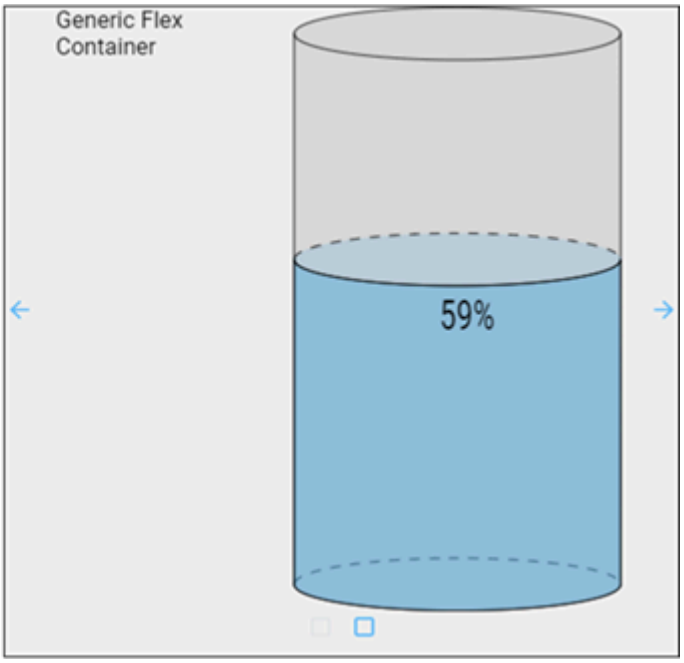
See the **Carousel Component Examples** page for more examples of using the Carousel component.

Carousel Component Examples

Example 1

In this example we made two view containers. One is a coordinate container and the other is flex container, named **generic_coordinate** and **generic_flex**, respectively. Each has a Label component and a Cylindrical Tank component. The Label's text property shows the name of the view it is in. The Cylindrical Tank's value property shows a level above 20. The generic_flex view is set to display as a row. We did not adjust any other settings on these two generic screens.



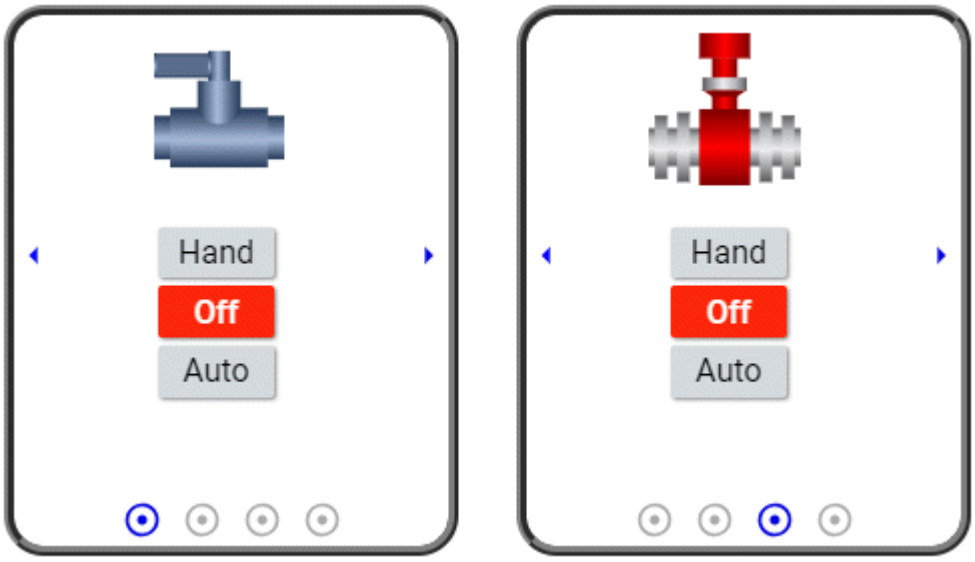


Once the two view containers are set up, we can create the Carousel with the following properties and values:

Property	Value	Style Category
props.views.0.viewPath	generic_coordinate	N/A
props.view.1.viewPath	generic_flex	N/A
props.autoplay.enabled	true	N/A
props.autoplay.pauseOnHover	true	N/A
props.appearance.useDefaultViewWidth	false	N/A
props.appearance.useDefaultViewHeight	false	N/A
props.appearance.dots.iconPath	/system/images/material/crop_square	N/A
props.appearance.arrows.enabled	true	N/A
props.style.borderStyle	solid	border
props.style.borderWidth	1px	border
props.style.backgroundColor	#EFEFEF	background

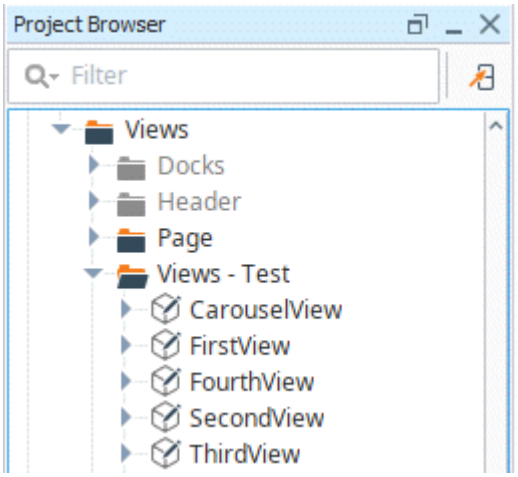
Example 2

In this example, we have set up a Carousel component that enables users to quickly move between four views showing valves and multistate buttons. The views must be created before the carousel, but it doesn't matter what the contents of the views are. You could use any combination of views including duplicates. Example 3 below describes setting up the views to scroll through in more detail.



They are named as follows in a **Views - Test** folder:

- CarouselView
- FirstView
- SecondView
- ThirdView
- FourthView

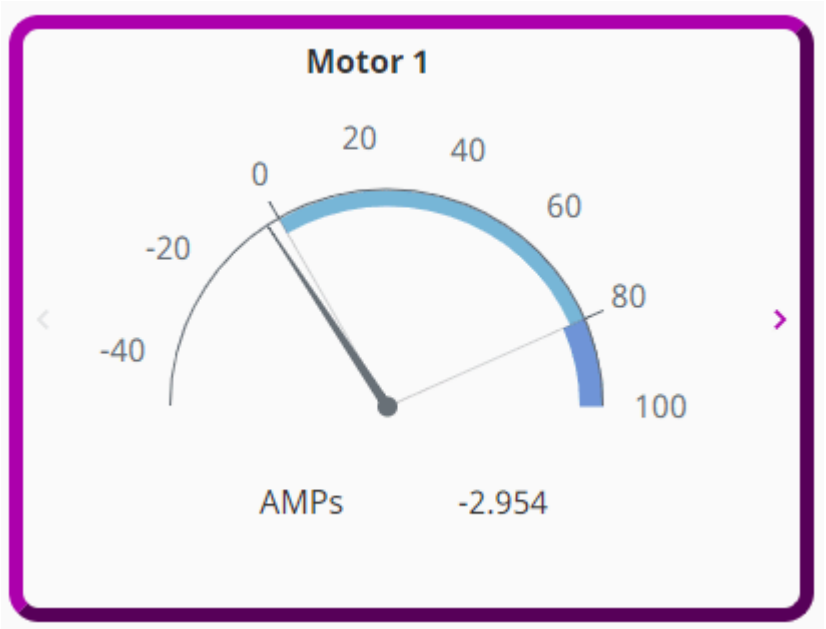


The properties used for the Carousel component are as follows:

Property	Value	Style Category
props.views.0.viewPath	Views - Test/FirstView	N/A
props.views.1.viewPath	Views - Test/SecondView	N/A
props.views.2.viewPath	Views - Test/ThirdView	N/A
props.views.3.viewPath	Views - Test/FourthView	N/A
props.autoplay.enabled	false	N/A
props.appearance.useDefaultViewWidth	true	N/A
props.appearance.useDefaultViewHeight	true	N/A
props.appearance.dots.enabled	true	N/A
props.appearance.dots.iconPath	material/adjust	N/A
props.appearance.dots.styles.active.fillColor	#0000D9	N/A
props.appearance.arrows.enabled	true	N/A
props.appearance.arrows.next.iconPath	material/arrow_right	N/A
props.appearance.arrows.next.fillColor	#0000D9	N/A
props.appearance.arrows.previous.iconPath	material/arrow_left	N/A
props.appearance.arrows.previous.fillColor	#0000D9	N/A
props.style.borderStyle	ridge	border
props.style.borderWidth	5px	border
props.style.borderRadius	20	border
props.style.borderColor	#808080	border

Example 3

This example demonstrates a more complex case of how to configure side scrolling using the Carousel component starting with the initial project setup.



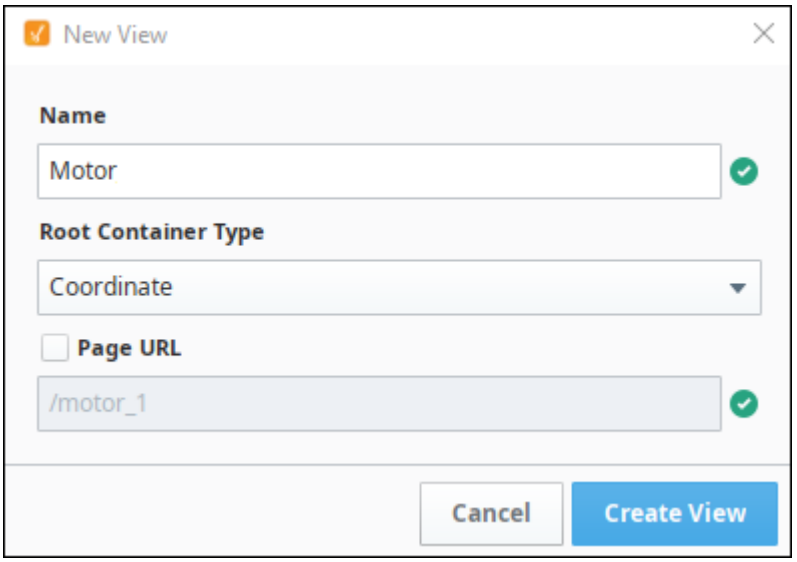
Initial Project Setup

First we'll create three views for the carousel to scroll through.

1. In the Project Browser, right-click on Views and select the **NewFolder** option. We named our folder **Carousel Example**.

2. Right-click on the **Carousel Example** folder and select the **NewView** option.

    - Name: **Motor**
    - Layout: **Coordinate**
    - Page URL: unchecked



3. Click **Create View**.

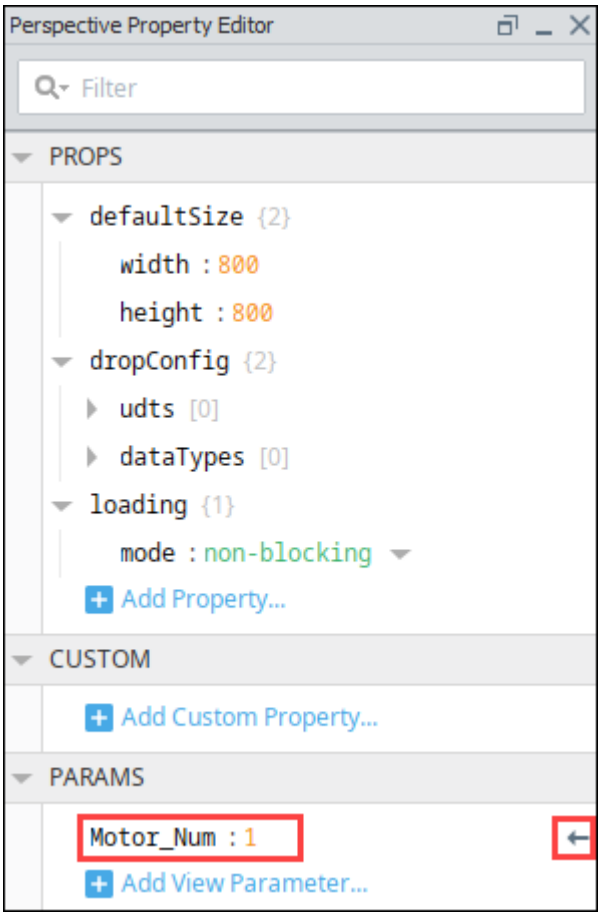
4. Next, create a view parameter that we can use for the motor number. In the Property Editor under PARAMS, click on the **Add View Parameter** icon.

    i. Select **value**.

    ii. Change key to **Motor_Num**.

    iii. Change value to **1**.

    iv. Make sure the arrow icon is facing to the left, indicting this is an input parameter.



5. Next, make a title that will change depending on the Motor that's being displayed. Drag a Label component onto the view.

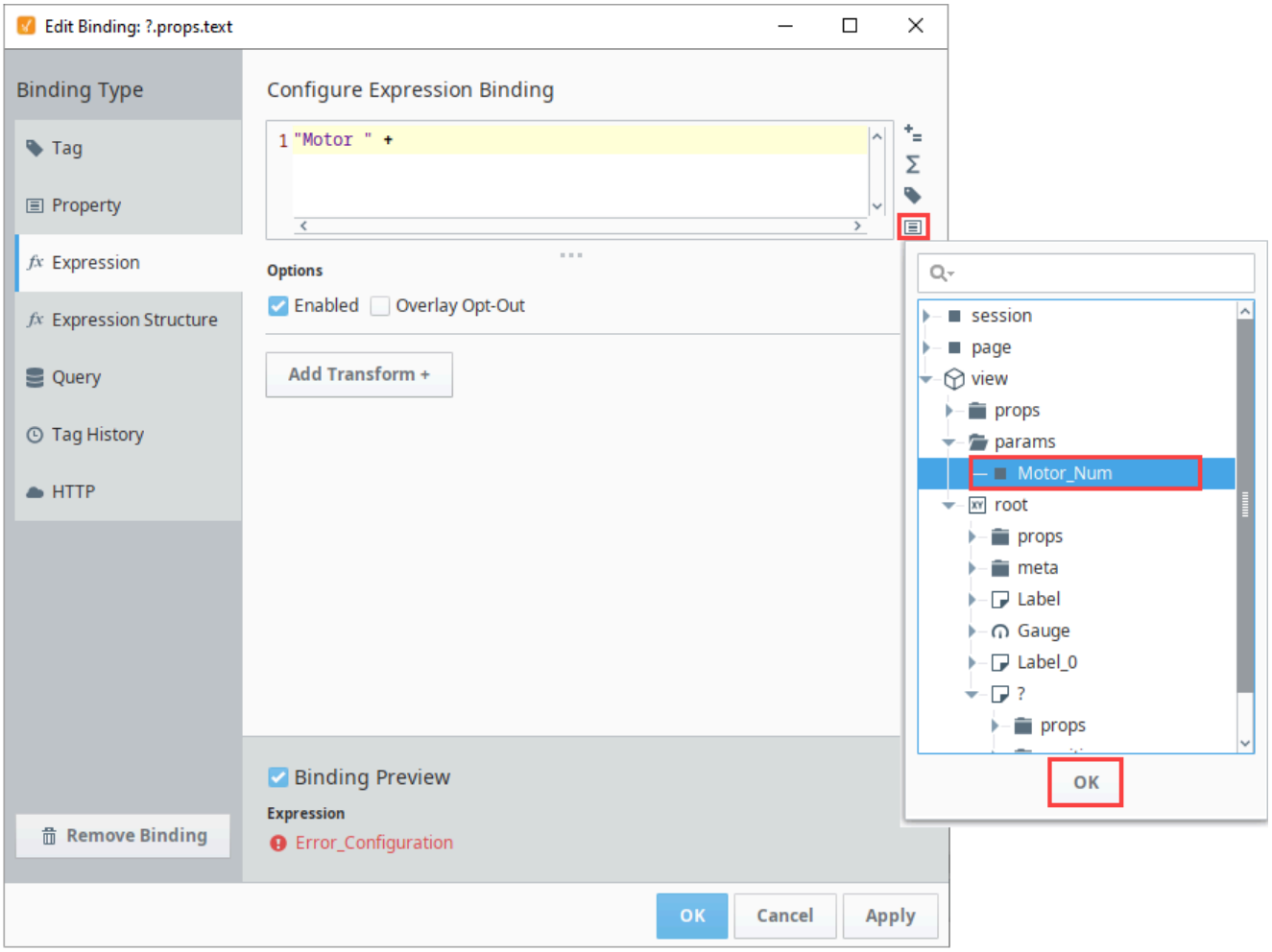
6. Bind the Label's text property to the view parameter as follows:

    i. Click the **binding** icon next to the text property.

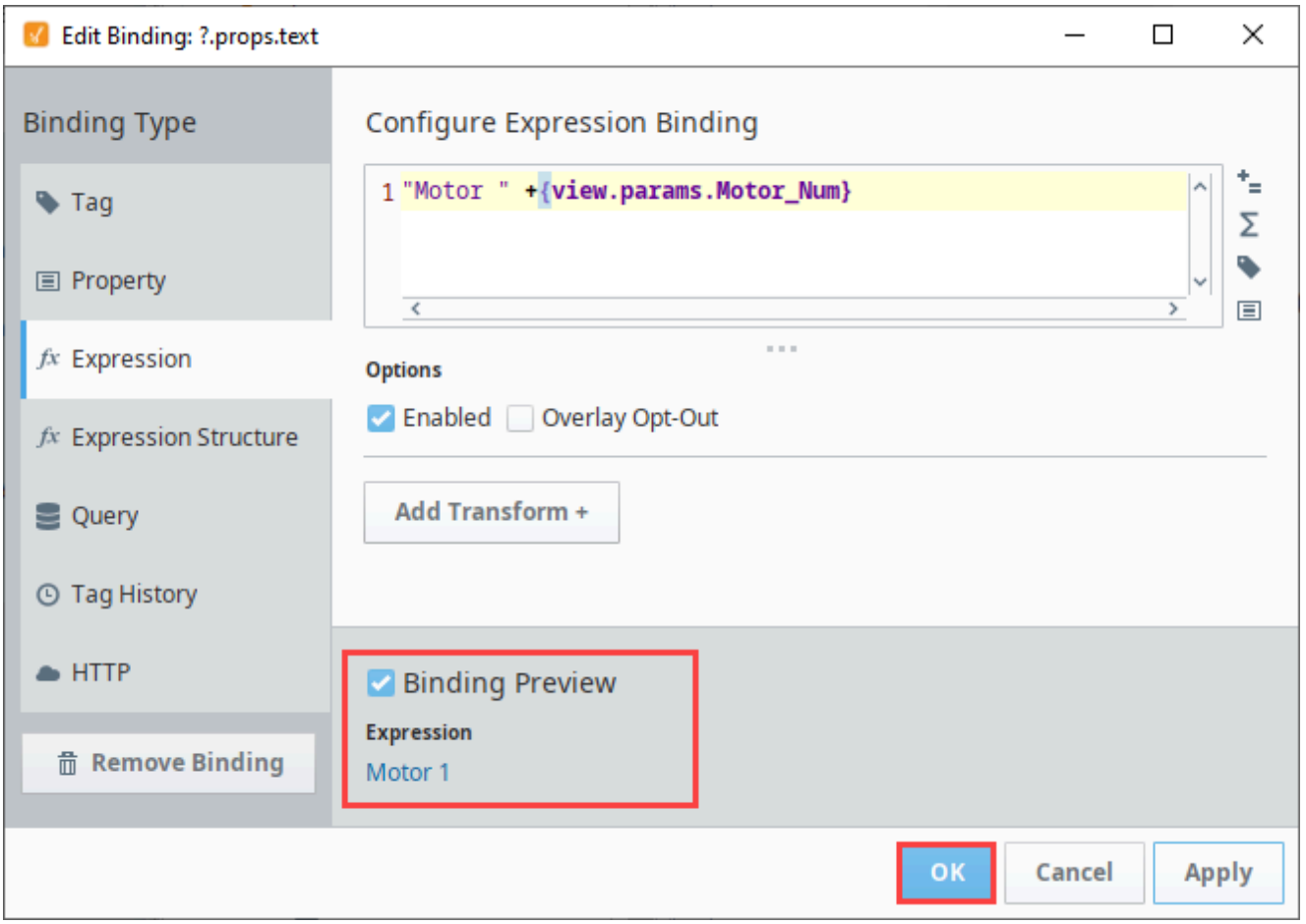
    ii. Select **Expression** as the binding type.

    iii. Enter "**Motor **" +.

    iv. Click the **Property** icon then scroll down and select the Motor_Num view parameter.



    v. Click **OK**. You'll now see the binding preview shows "Motor 1."



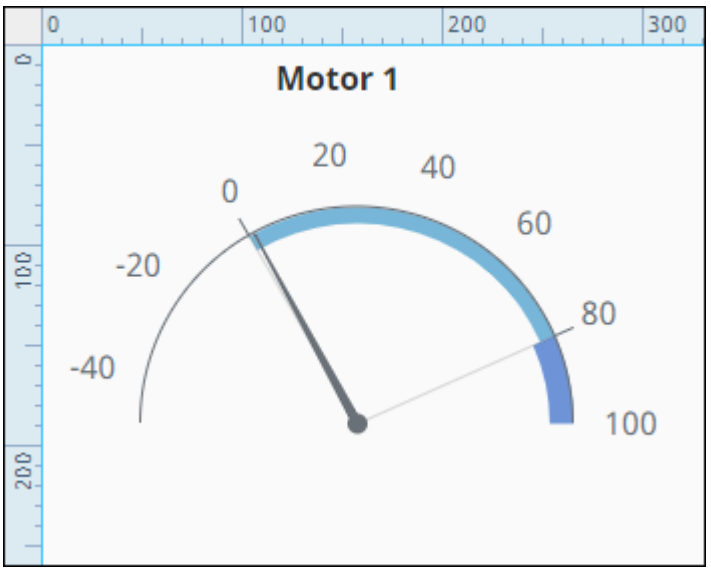
vi. Click **OK** to save the binding.

7. Next, drag a Gauge component onto the view to display motor Amps.

8. Align the Motor label centered above the Gauge component.

9. In the Property Editor, expand the outerAxis properties. Change the properties as follows:

   - minValue: **-50**
   - maxValue: **100**



10. Now we'll set up an indirect tag binding using one of the motor tags in the Dairy simulator. (For more information on the Dairy simulator, see **Programmable Device Simulator**.) Select the Gauge component.

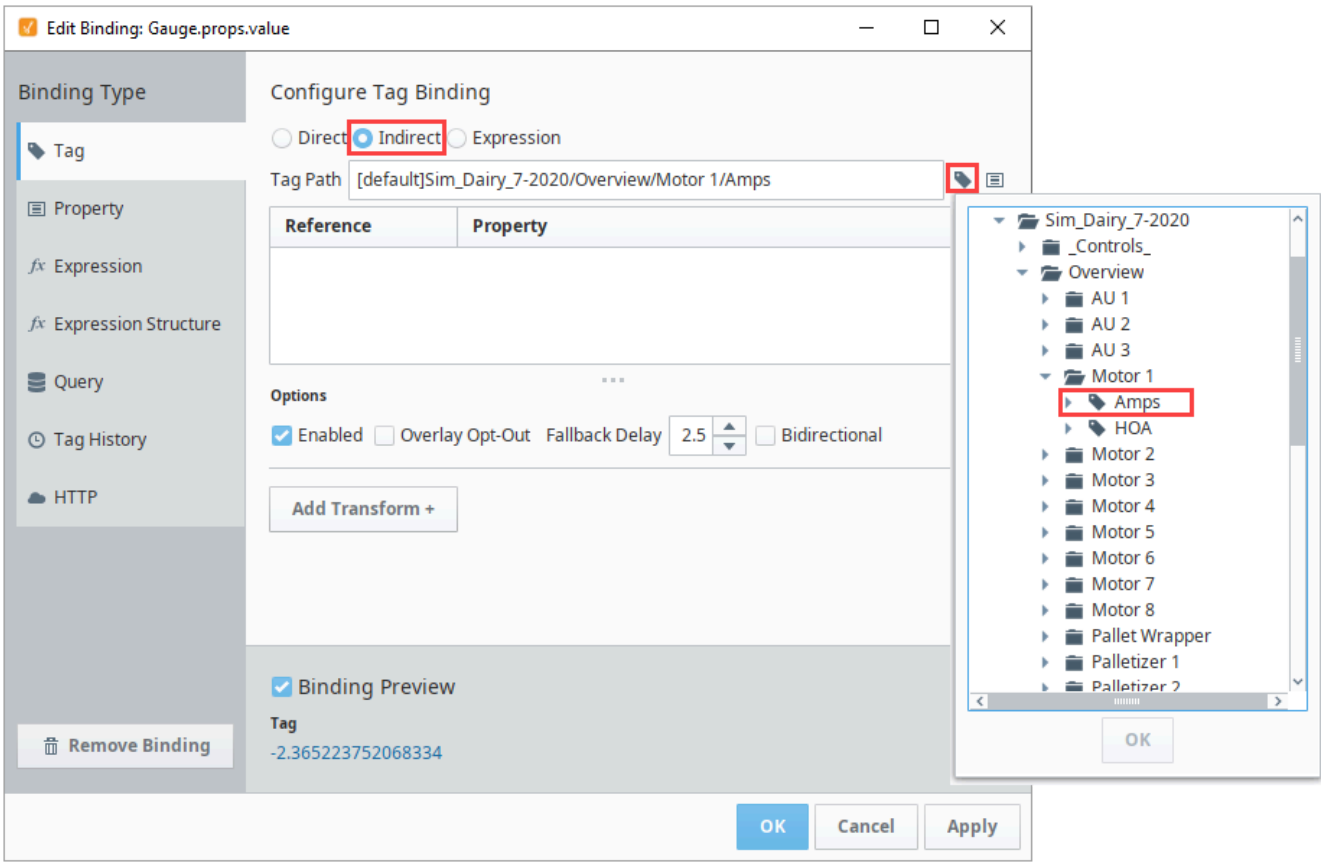
11. In the Property Editor, click the **binding** icon next to the value property. On the Configure Tag Binding page set the following:

    i. Choose **Tag** as the binding type.

    ii. Select the **Indirect** option.

    iii. Next to the Tag Path field, click on the **Tag** icon and navigate to the the **Motor 1/Amps** tag in the **Dairy simulator program**.

    iv. Click **OK**.



    v. In the Tag Path field, replace the **1** with **1**.

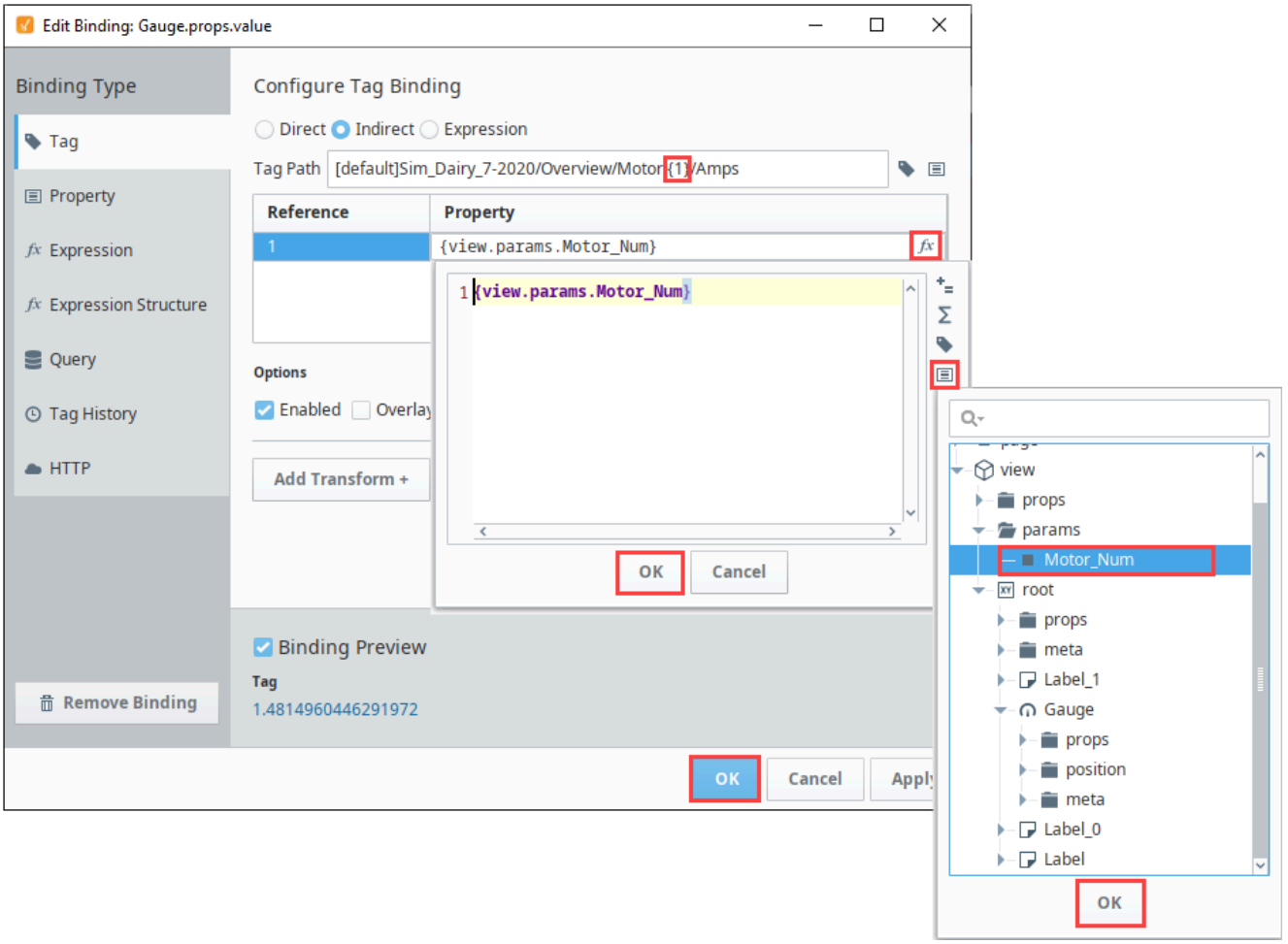
    vi. In the References list, under Property, click on the **Functions** icon.

    vii. Click the **Properties** icon.

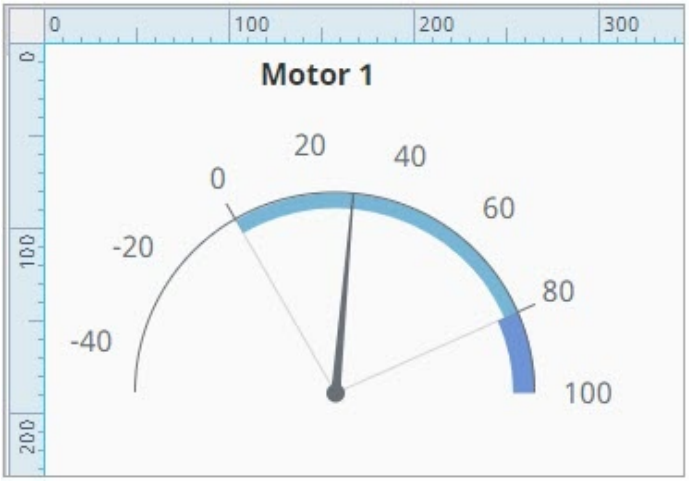
    viii. Scroll to the view params folder and select **Motor_Num**. Click **OK**.

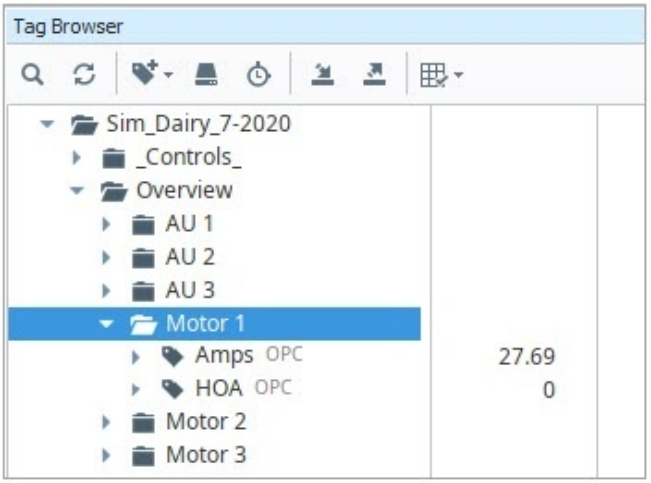
    ix. Click **OK** again. You'll see the binding in the preview area.

x. Click **OK** to save the binding.



12. Confirm the Gauge now displays the value of the Amps tag.





13. Drag a Label component onto your view. Place it under the Gauge component and change the text property to **AMPS**.

14. Drag another Label component onto your view and place it next to the AMPS label. We'll set up a similar indirect tag binding on this label.

15. Select the second Label click the **binding** 🔗 icon next to the value property.

i. Select **Tag** as the binding type and click the **Indirect** radio button.

ii. Next to the Tag Path field, click on the **Tag** icon and navigate to the the Motor 1/Amps Tag in the Dairy simulator program.

iii. Click **OK**.

iv. In the Tag Path field, replace the **1** with **1**.

v. In the References list, under Property, click on the **Functions** icon.

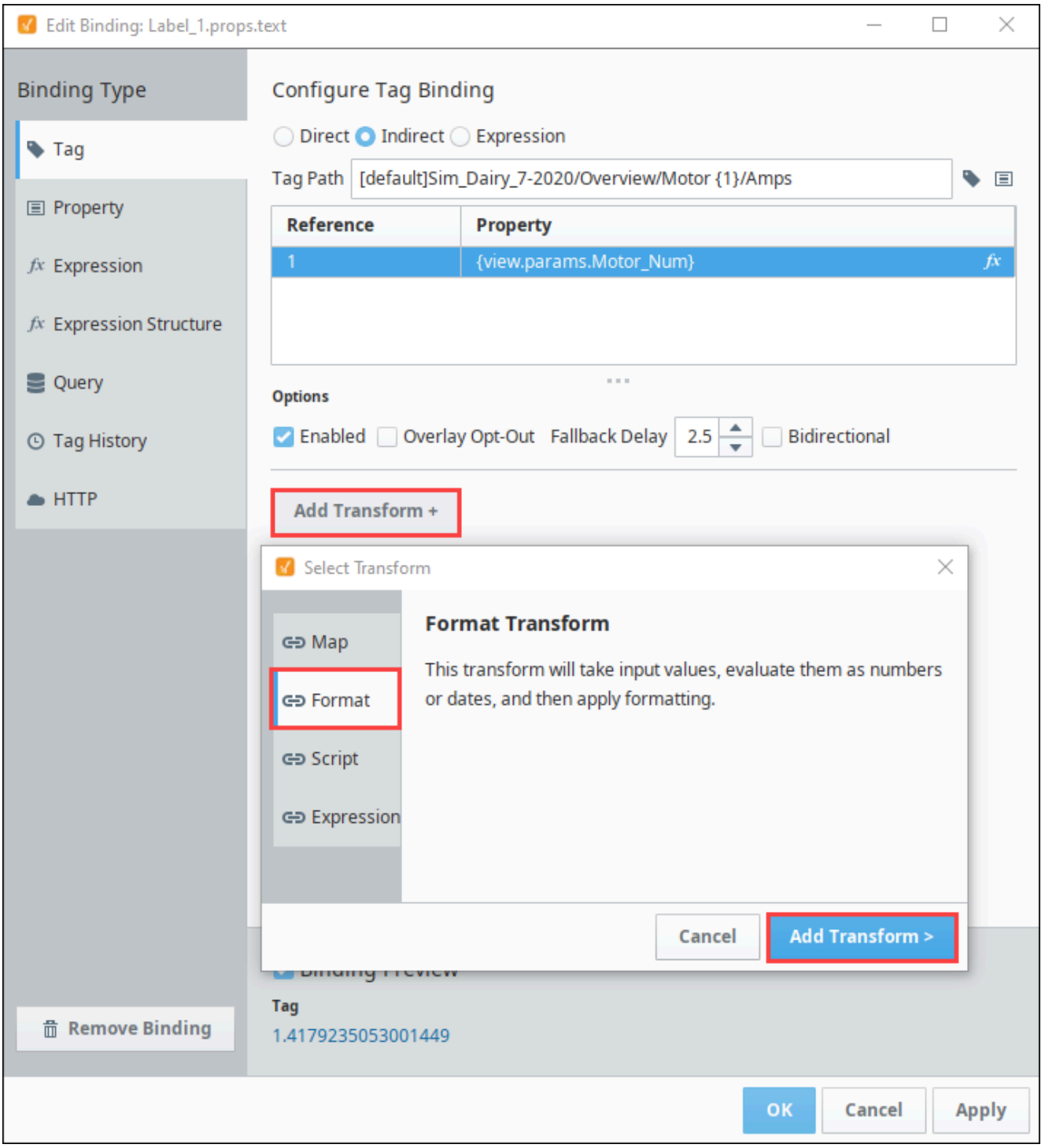
vi. Click the **Properties** icon.

vii. Scroll to the view params folder and select **Motor_Num**. Click **OK**.

viii. Click **Apply**. You'll see the binding in the preview area.

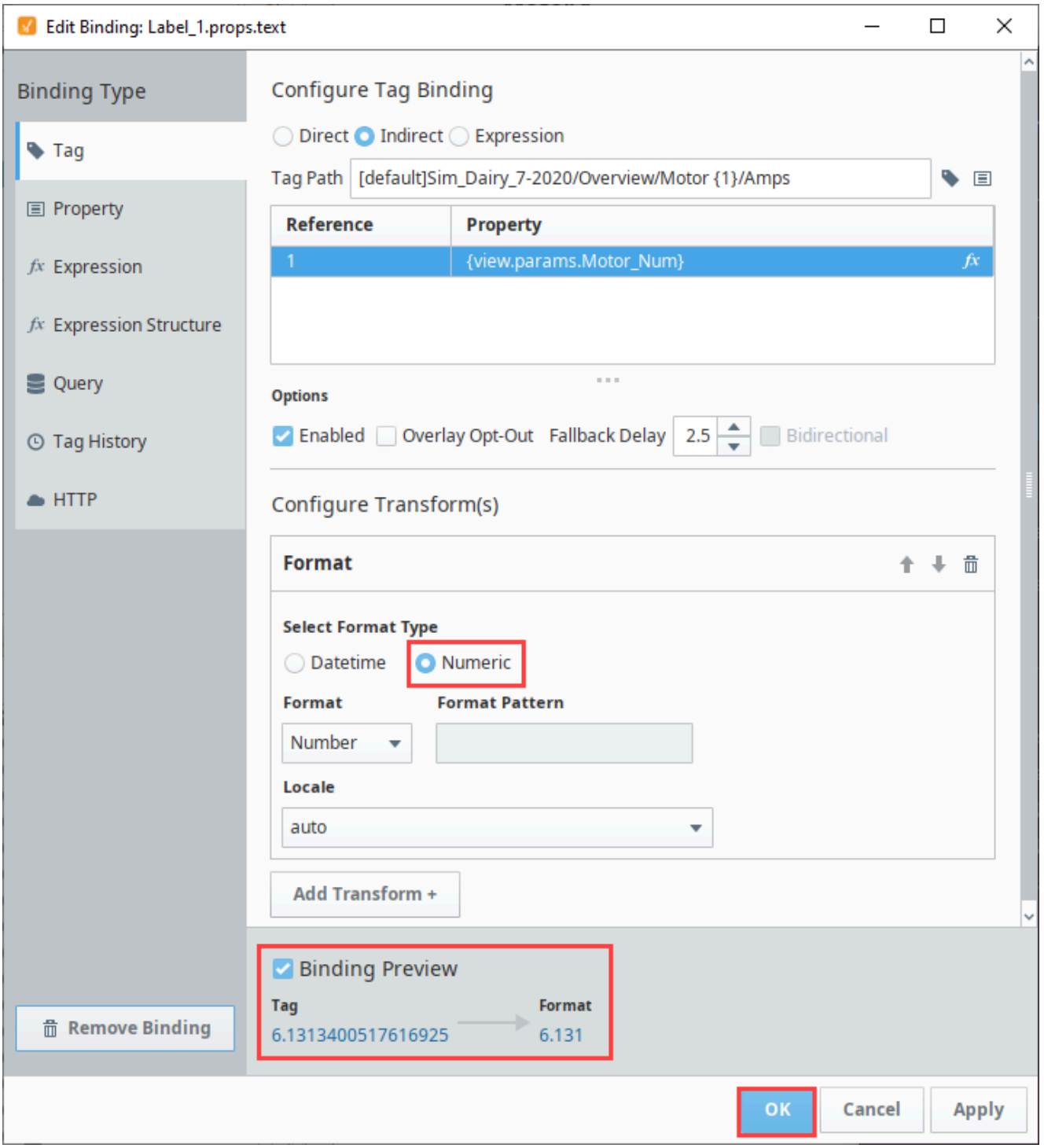
ix. For this Label component we don't want the full tag value displayed. So we'll add a Transform to limit it to two decimal points. Click on **Add Transform**.

x. Select **Format**, then click **Add Transform**.



xi. Select **Numeric** as the Format type. The displayed value will now be shortened to just two decimal points. You can see the format in the Binding Preview.

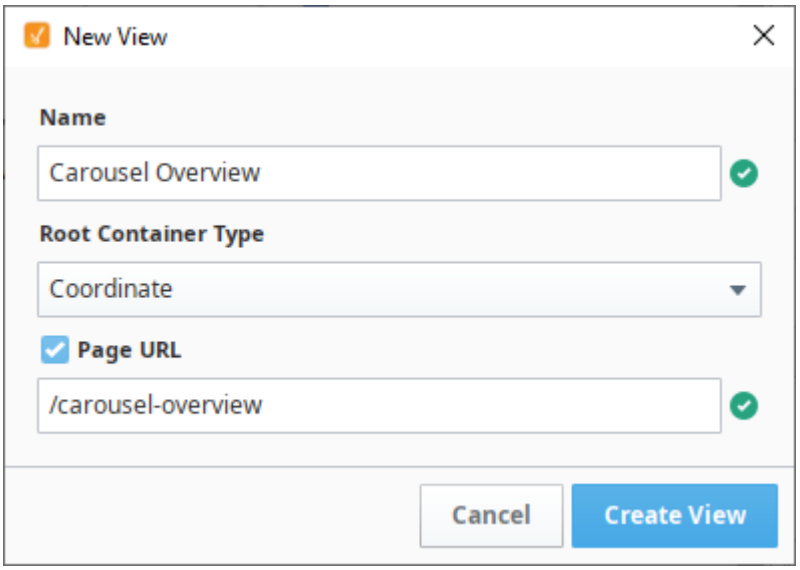
xii. Click **OK** to save the binding.



Set Up the Carousel View

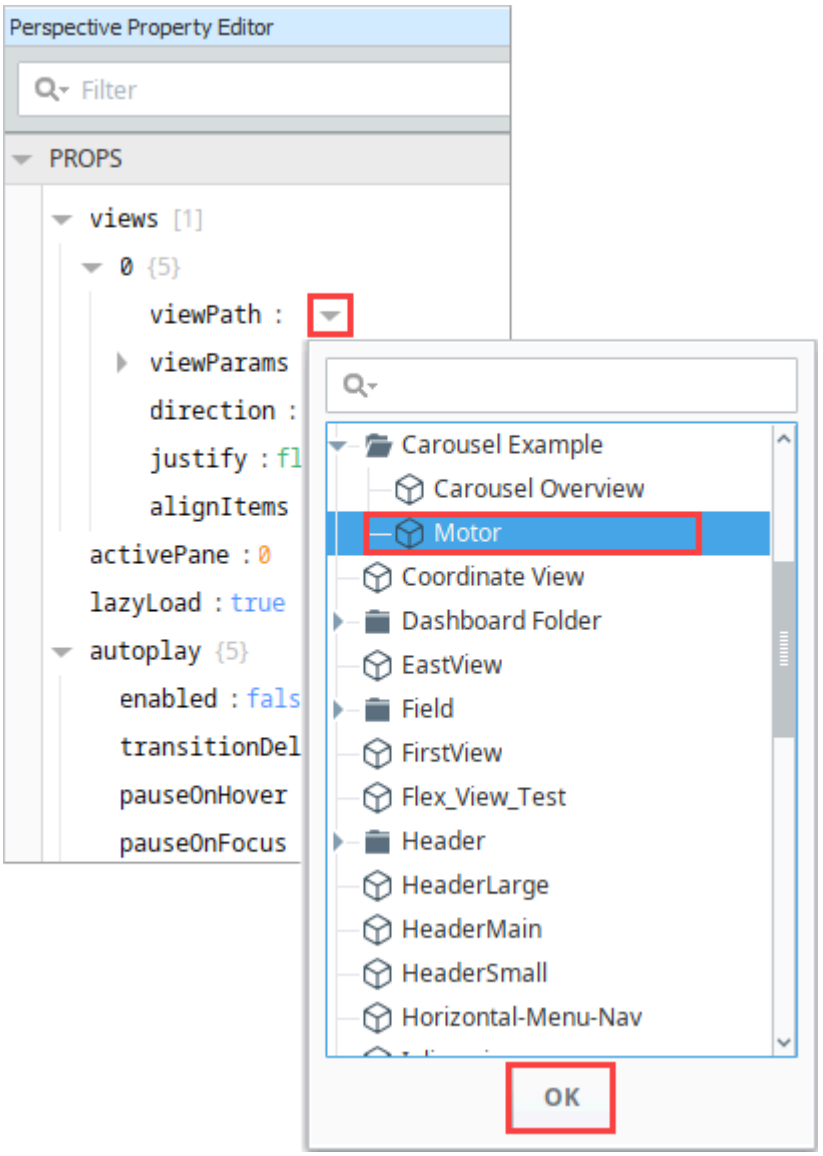
Now we'll set up a view that holds the Carousel component.

1. Right-click on the Carousel Example folder and select the **NewView** option.
2. Name the view **Carousel Overview**. Check the Page URL option.
3. Click **Create View**.



4. Drag a Carousel component onto the view.

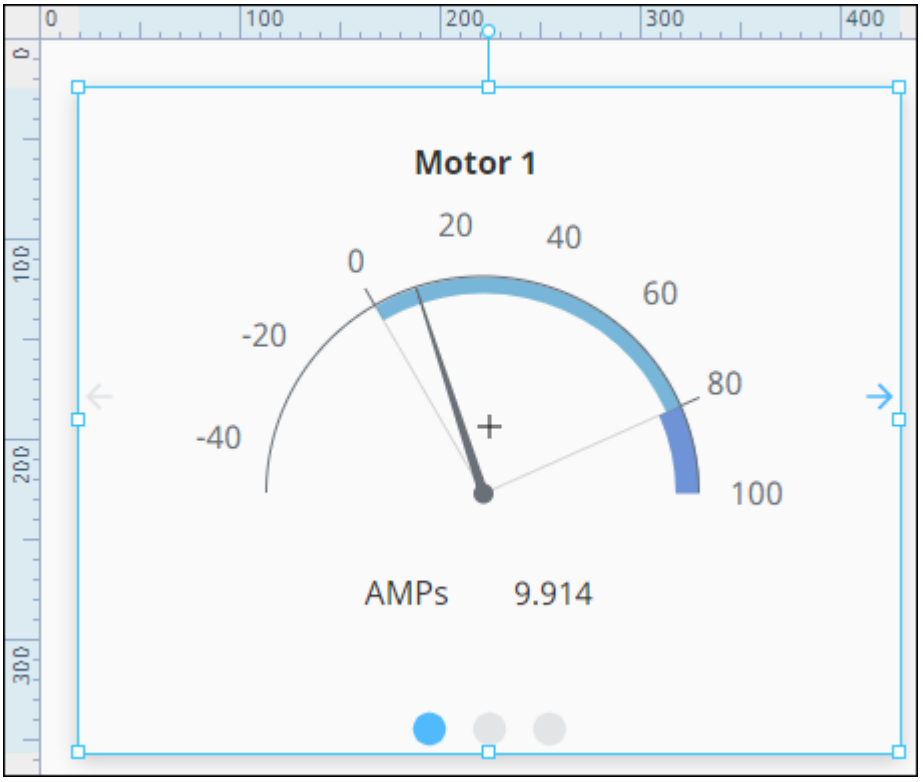
5. In the Property Editor, click the **Expand** ▼ icon in expand the props.views.0.viewPath property and select the **Motor** view.



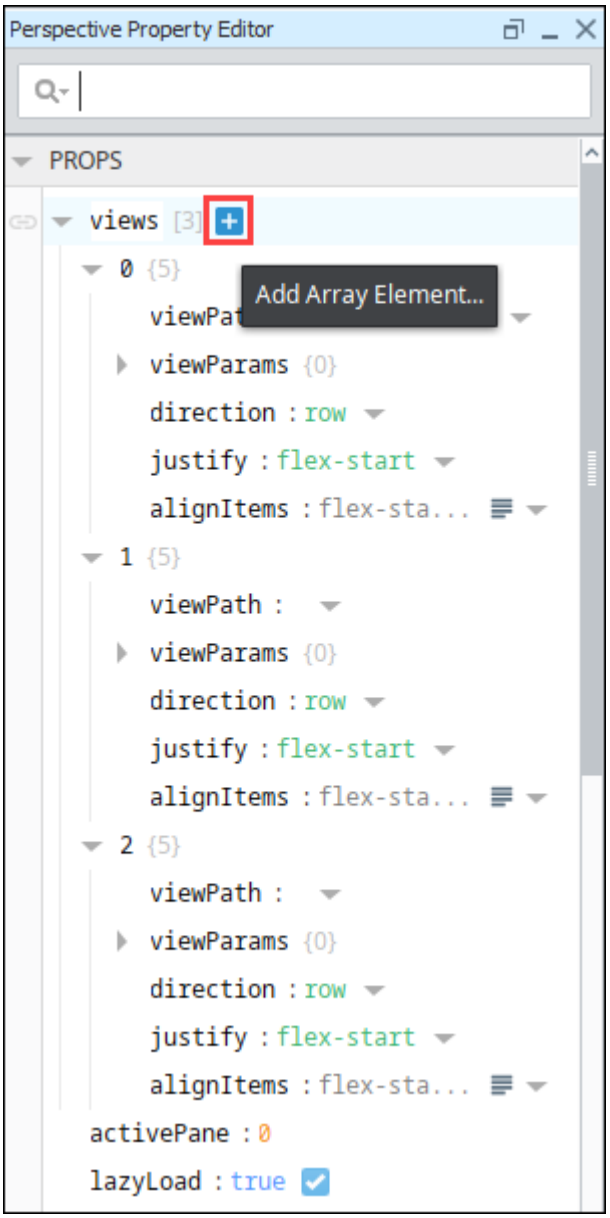
6. Under viewParams, click the **Add** ⊞ icon then choose **value**.

7. Replace key with **Motor_Num** and replace value with **1**.

8. Click **OK**. The Motor view now appears in the Carousel component. You may need to expand your Carousel component slightly to fit the Motor view in without scrollbars.



9. Next, we'll add two more views. Click on the **Add** ➕ icon two times for the Property Editor's views property.



10. Under the props.views.1.viewPath property, choose the **Motor** view.

    i. Under viewParams, click the **Add** icon then choose value.

    ii. Replace key with **Motor_Num** and replace value with **2**. This will point this instance to the Motor 2/AMPS Tag.

    iii. Click **OK**.

11. Under the props.views.2.viewPath property, choose the **Motor** view.

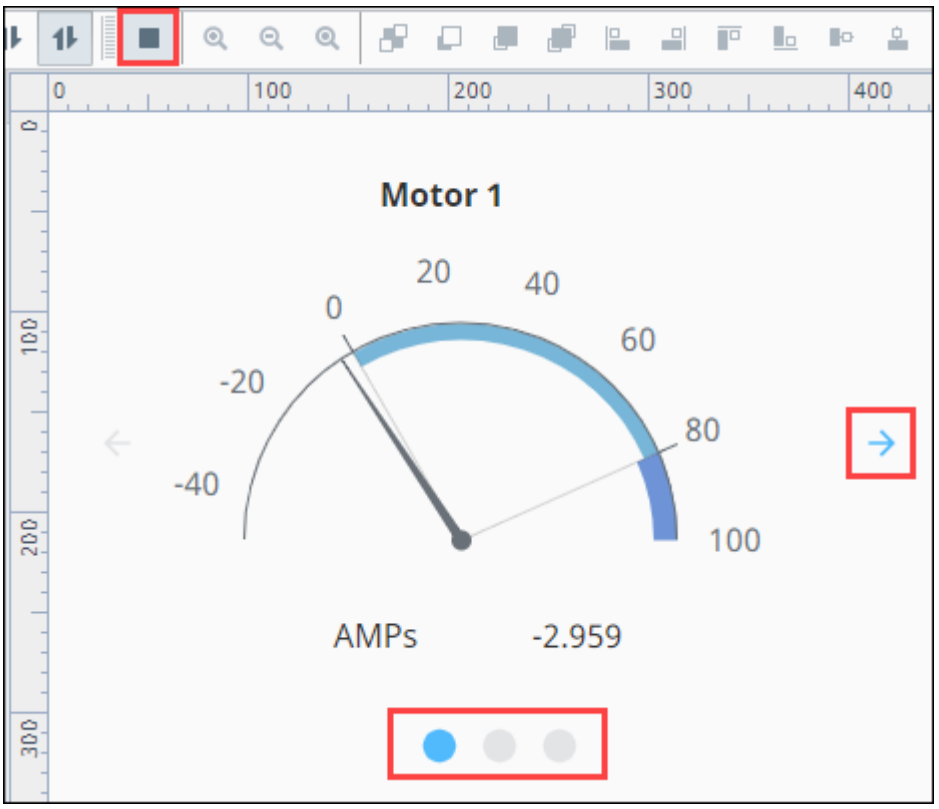
    i. Under viewParams, click the **Add** icon then choose value.

    ii. Replace key with **Motor_Num** and replace value with **3**. This will point this instance to the Motor 3/AMPS Tag.

    iii. Click **OK**.

12. Save your project.

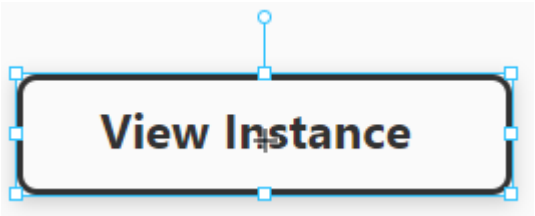
13. Put the Designer into Preview mode. Click the left and right arrows or the dots to scroll between the three Motor views.



14. Lastly, we decided to change a few properties on the Carousel to update the appearance. Here are the settings we used:

Property	Value
props.appearance.dots.enabled	false
props.appearance.arrows.next.iconPath	material/navigate_next
props.appearance.arrows.next.fillColor	#AC00AC
props.appearance.arrows.previous.iconPath	material/navigate_before
props.appearance.arrows.previous.fillColor	#AC00AC
props.style.borderStyle	outset
props.style.borderColor	#AC00AC
props.style.borderWidth	7
props.style.borderTopLeftRadius	15
props.style.borderTopRightRadius	15
props.style.borderBottomLeftRadius	15
props.style.borderBottomRightRadius	15

Perspective - Embedded View



Component Palette Icon:



Description

The Embedded View component allows you to include an entire view inside another. Using this component allows you to select a view to display, and to pass parameters into the view. Because of this, views can easily act as templates for information.

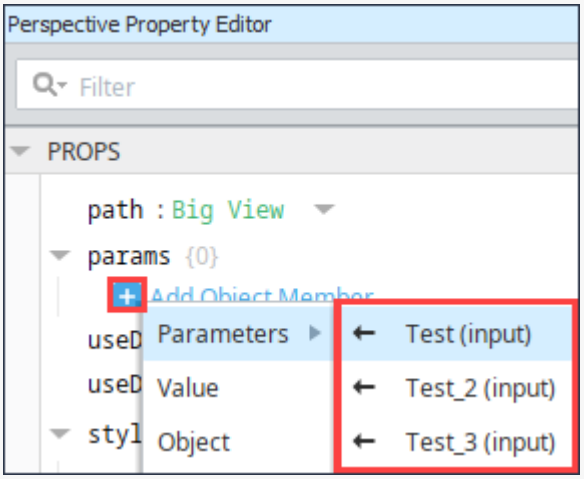
For example, you could create a tank view and embed several into another, larger view that shows an overview of the facility.

Using an Embedded View is different than a container because you cannot alter the contents of the view. A new container would allow you to create a new grouped set of components.

> **SLOW PERFORMANCE**
>
> Embedded views are convenient, but oversaturating your view may come with performance issues. Too many embedded views in a single view or too many nested embedded views, along with potential race conditions from passing parameters between embedded views may cause slow performance.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

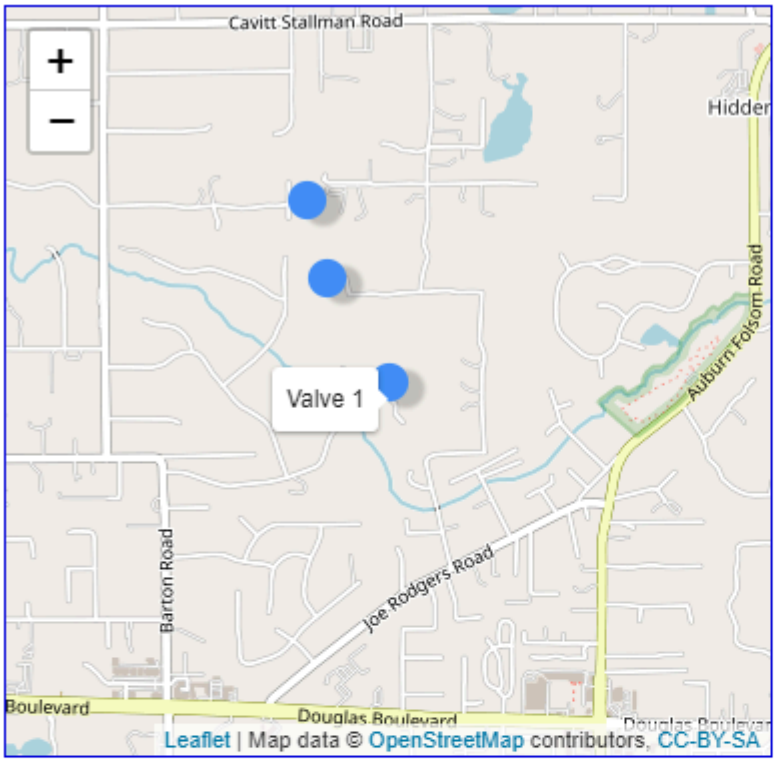
Name	Description	Property Type
path	Path of the view to load in wrapper. If a path is present in the path property field, an Open View icon will appear that will navigate directly to the view when clicked.	value: string
params	Parameters for the view. If passing parameters into the embedded view, the names here must match the parameters on that view. A dropdown list of parameters is available when the user clicks the **Add Object Member** icon. This makes it easy to add parameters from the embedded view. See also **Embedded Views**. 	object
useDefaultViewWidth	Use of view's default width instead of adjusting based on the content's width.	value: boolean
useDefaultViewHeight	Use of view's default height instead of adjusting based on the content's width.	value: boolean
loading	View loading settings. • **order**: Controls when the embedded view loads: alongside the parent view loading (`with-parent`), or after the parent view finishes loading (`after-parent`). Default is with-parent. Generally speaking, `with-parent` is more efficient for the browser, but in some cases can feel slower overall. Alternatively, `after-parent` is generally less efficient for the browser and can add to the overall	object

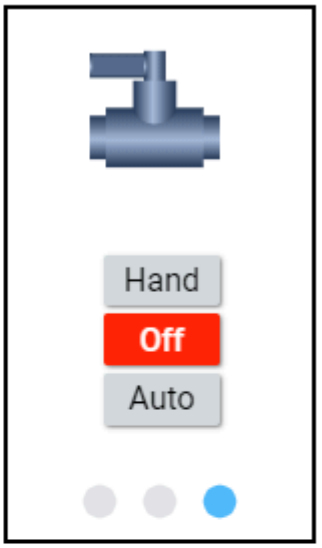
Name	Description	Property Type
	load time. However, since it allows the parent view to load first, `after-parent` may feel quicker since the topmost layer of views get started up sooner. Value is boolean.	
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

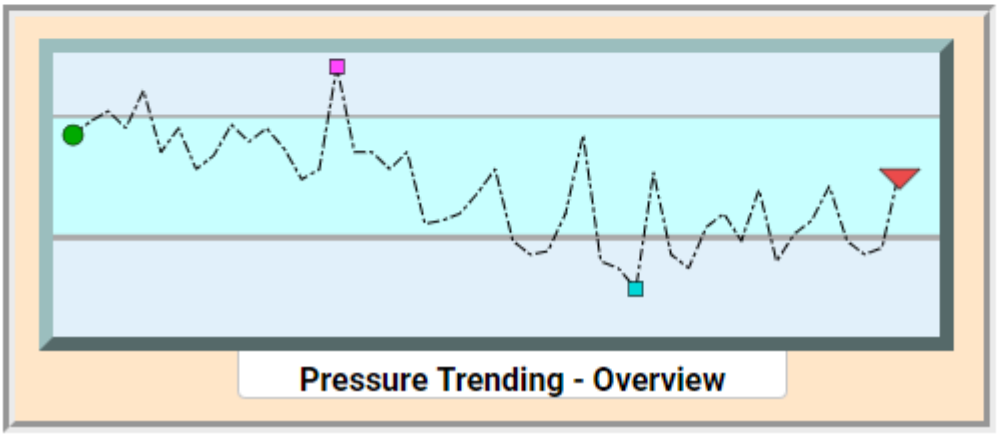
The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example





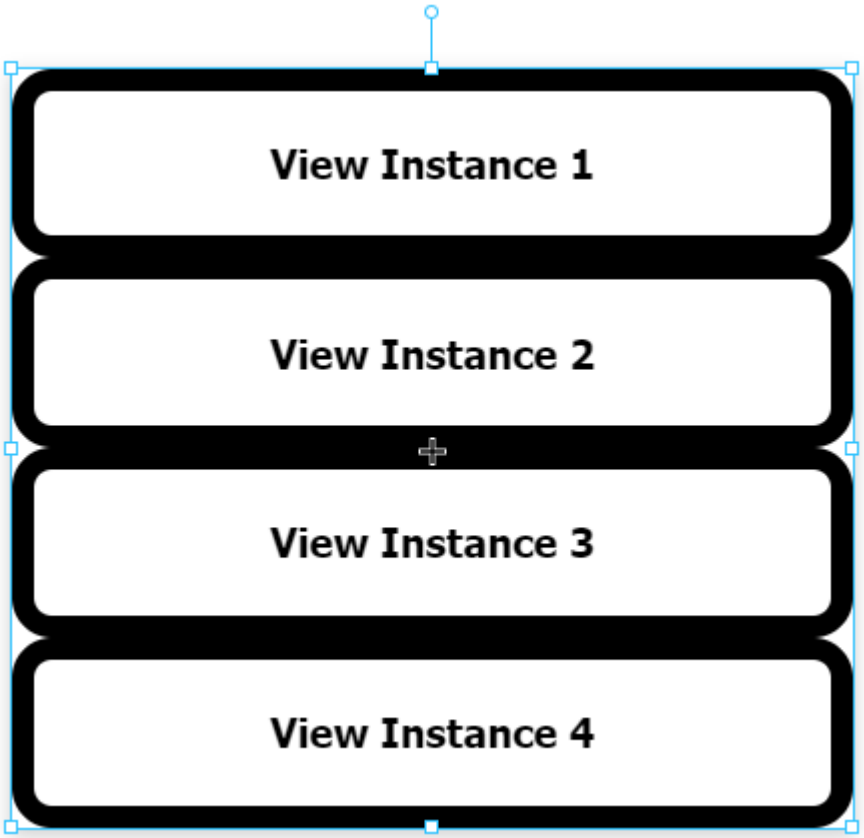
Embedded View



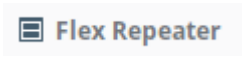
In this example, there is a Map component and a Carousel component on the top of the page. Underneath them, we've placed an Embedded View component. The idea was to embed this overview to give users a quick visual reference to bigger picture trends for the site. This example assumes you have a view already created named "Pressure Trend Overview". The table below shows the properties for just the Embedded View.

Property	Value	Style Category
props.path	Pressure Trend Overview	N/A
props.style.backgroundColor	#FFE8CC	background
props.style.borderStyle	groove	border
props.style.borderWidth	6px	border

Perspective - Flex Repeater



Component Palette Icon:



Description

The Flex Repeater component lets you easily create multiple instances of views for display in another view.

When first dropped on a view, the Flex Repeater looks like any other empty container. Set the **path** to the component that you want to create multiple instances of, and then under **instances** add an object for each instance that you want to create. The **object** will usually contain one or more parameters, including the instance's own **index**, that will be passed into that particular instance. As a side note, overwriting the index parameter is not recommended.

The flex repeater is functionally very similar to the **Flex Container component**. Both components are based off of the **CSS flexbox**, and both abide by similar rules in regards to how child objects are positions. However the Flex Repeater differs from the contain in two notable ways:

1. The Flex Repeater can only have embedded views as direct children, where as the Flex Container can have any type of component.
2. The Flex Repeater can create instances of views from the runtime, by adding additional elements to the **instances** array. The flex container does not have anything resembling this functionality.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

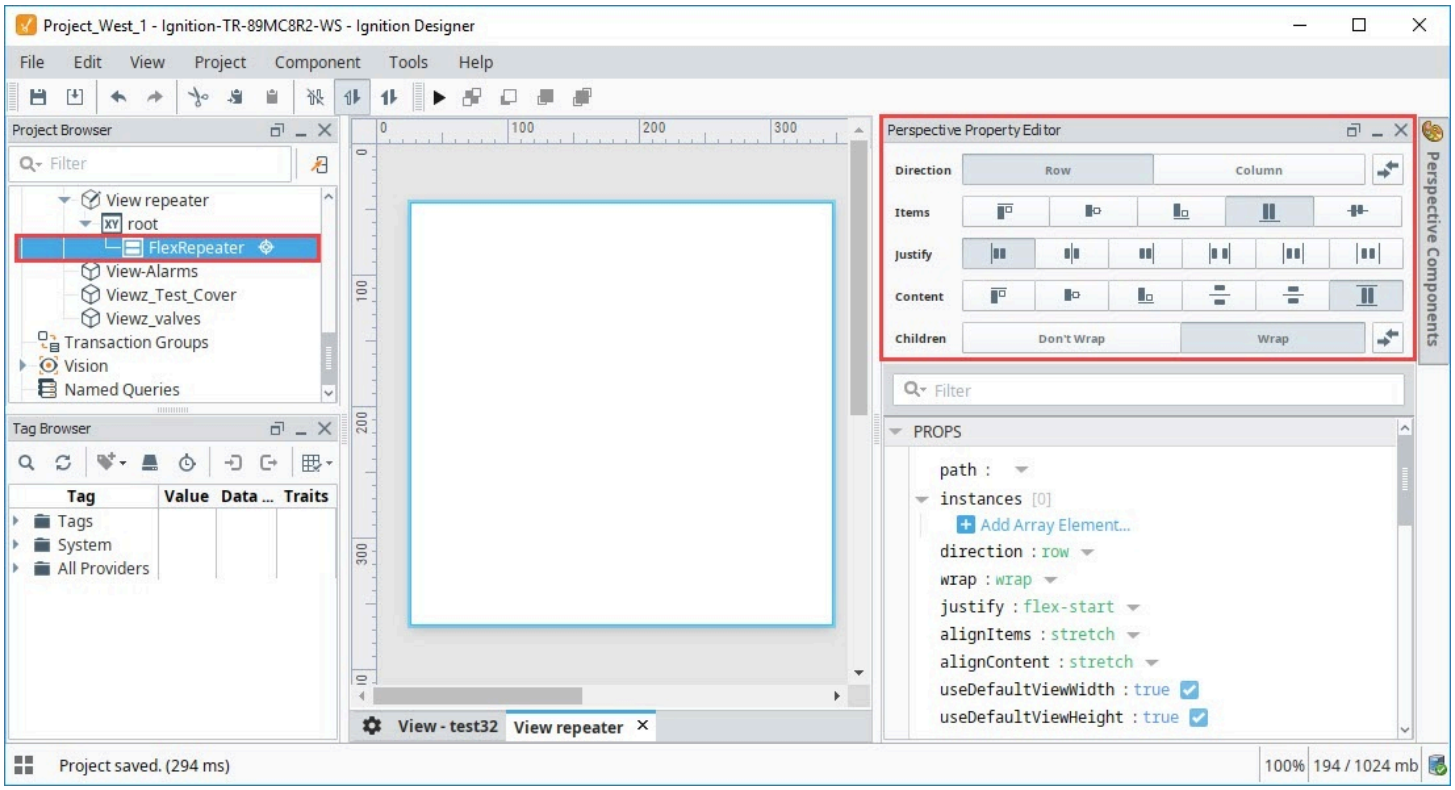
Name	Description	Property Type
path	Path to the desired view to display. If a path is present in the path property field, an Open View  icon will appear that will navigate directly to the view when clicked.	value: string
instances	Number of instances of the view that you want to display in the container. Each instance can contain an instanceStyle and instancePosition property. Changing these properties will override any styles and position settings applied by elementStyle and elementPosition. This is where a value property can be added that matches up with a parameter in the view to pass in a value. • **instanceStyle**: Sets a style for this instance of a view. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **instancePosition**: Position properties such as grow, shrink, or basis that would apply to all instances. A dropdown list of parameters is available when the view being displayed has view parameters. This makes it easy to add, delete, or synchronize parameters from that view.	array
direction	Direction of layout of repeated views. Options are row, row-reverse, column and column-reverse.	value: string
wrap	Whether the container should allow instances to wrap to the next line if space has run out. Options are nowrap, wrap, wrap-reverse.	value: string
justify	Adjusts placement of instances along the main axis when there is extra space, which may be used to fill areas before, after, or in-between: flex-start, flex-end, center, space-between, space-around, space-evenly.	value: string

Name	Description	Property Type
alignItems	Adjusts placement of instances along the cross axis when there is extra space: flex-start, flex-end, center, baseline, stretch.	value: string
alignContent	Adjusts alignment of wrapped content when there is free space in the cross axis: flex-start, flex-end, center, space-between, space-around, stretch.	value: string
useDefaultViewWidth	Use view's default width instead of adjusting based on the content's width.	value: boolean
useDefaultViewHeight	Use view's default height instead of adjusting based on the content's height.	value: boolean
elementStyle	Sets a style for this element. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
elementPosition	Sets a position for this element. Element position properties that apply to all instances, unless overridden by instancePosition • **grow**: **grow** and **shrink** control the way that a component responds to changes in the flex container's width or height. For columns **row** controls what happens when additional space is available. Value is numeric. • **shrink**: **shrink** controls what happens when the component does not have enough space to fulfill its basis. Value is numeric. • **basis**: Controls the default size of a component along the flex repeater's direction. You can enter the value in pixels (e.g. 75px), as a percentage of the total length of the container (e.g. 50%), or you can use auto. All components configured to auto will equally share the available space in the container. Value is numeric.	object

Name	Description	Property Type
loading	View loading settings. • **order**: Controls when the embedded view loads: alongside the parent view loading (`with-parent`), or after the parent view finishes loading (`after-parent`). Default is with-parent. Generally speaking, `with-parent` is more efficient for the browser, but in some cases can feel slower overall. Alternatively, `after-parent` is generally less efficient for the browser and can add to the overall load time. However, since it allows the parent view to load first, `after-parent` may feel quicker since the topmost layer of views get started up sooner. Value is boolean.	object
style	Sets a style for the Flex Repeater. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **classes**: Sets a **style class** for the Flex Repeater. • **overflow**: Options are auto, visible, scroll, or hidden. Value is string.	object

User Interface

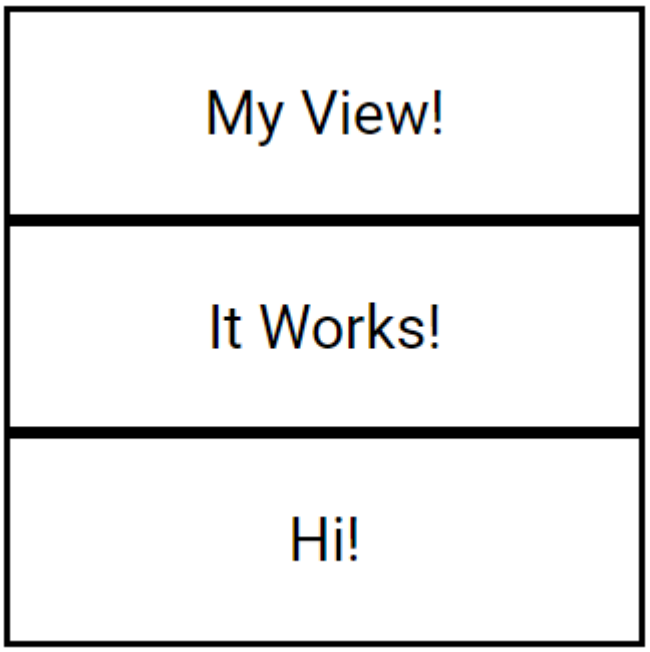
When a Flex Repeater is deep selected, there is an interface at the top of the Perspective Property Editor that enables you to set the container's properties. Functionality is similar to that of the Flex Container component. See **Perspective - Flex Container**.



Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example



1. In order to use the Flex Repeater, you need a view that will be shown inside the repeater. To do this, we made a new View with a Coordinate layout called "RepeatedView".

    i. In the Project Browser, select the new View
    ii. In the Property Editor. Under **PARAMS**, select **Add View Parameter**, select a property type of **Value**.
    iii. Rename the parameter from "key" to "labelText".
    iv. Select the **root** container for the view, and set the **mode** property to "percent".

2. Add a Label component to the view, and configure a property binding on its **text** property to the **labelText** parameter we just created. We set the component's **alignVertical** property to "center", and we stretched the label to fill the entire view. We also configured some styling on the Label:

Style Category	Value
borderStyle	solid
fontSize	30px
textAlign	center

3. Now we can configure our Flex Repeater. Drag a Flex Repeater component onto the view, then set the following properties:

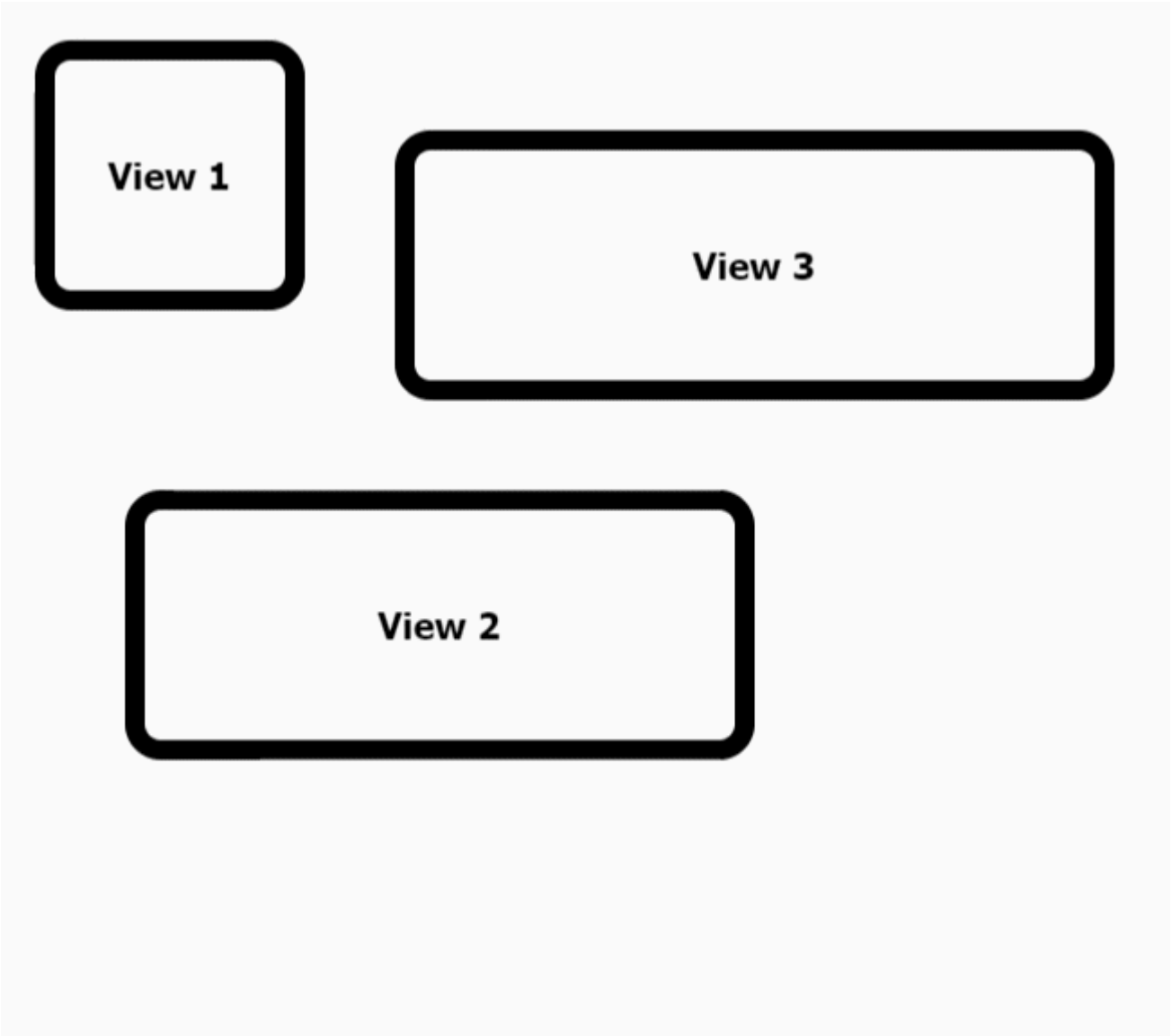
Style Category	Value
path	RepeatedView
direction	Column
useDefaultViewWidth	false
useDefaultViewHeight	false

4. Finally, create three **object** members in the **instances** array. Add a labelText property to each object of type **value**, and replace the value strings to the desired strings to show. Here's how the property editor looks for our Flex Repeater:

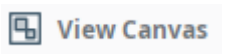


5. You should now see the Flex repeater populated as shown in the image above.

Perspective - View Canvas



Component Palette Icon:



Description

The View Canvas component can display multiple Perspective views, each positioned on a coordinate based system. The component offers smooth transition animations when views are relocated. Familiarity with CSS is helpful in taking full advantage of the this component.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
instances	Array of views to display in the canvas. • **position**: Mode that defines how the element is positioned within the canvas. For position absolute, the view is placed within the canvas based on its top, left, bottom, right positions. Views that are positioned absolute do not participate in the flow of the document. For position relative, the view is placed within the canvas placed in the normal document flow and then offset by its top, left values. This is the same with left and right.Options are relative or absolute. Default is absolute. Value is string. • **top**: The top position of the view. Value is numeric. • **left**: The left position of the view. Value is numeric. • **bottom**: The bottom position of the view. Note: If both top and bottom are set, bottom is respected only if position is set to absolute and height is unspecified. Value is numeric. • **right**: The right position of the view. Note: If both left and right are set, left is respected only if position is set to absolute and width is unspecified. Value is numeric. • **zIndex**: The z-order position of the view. Value is numeric. • **width**: The width of the view. Value is numeric. • **height**: The height of the view. Value is numeric. • **viewPath**: Path to the view you want to display. If a path is present in the viewPath property field, an Open View icon will appear that will navigate directly to the view when clicked. Value is string. • **viewParams**: The parameters of the view. A dropdown list of parameters is available when the user clicks the **AddObject Member** icon. This makes it easy to add parameters from the rendered view. • **style**: Sets a style for this view. Full menu of **style options** is available for text, background, margin and	array

Name	Description	Property Type
	padding, border, shape and miscellaneous. You can also specify a **style class**.	
transitionSettings	Transition settings on each view. The properties affected by transition settings are top, left, bottom, right, and zIndex. • **duration**: Duration of the transition. Units are seconds or milliseconds. Value is numeric. • **timingFunction**: Mathematical function that defines how fast one-dimensional values change during the transition. The transition can be described as a cubic Bezier or steps function. The presets for cubic Bezier functions are linear, ease, ease-in, ease-in-out, and ease-out. The presets for steps functions are step-start and step-end. Value is string.	value: numeric
enableTransitions	Determines whether transitions should play when transitions are defined.	value: boolean
defaultStyle	Sets a style for all views. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
style	Sets a style for the canvas. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Scripting

See the **Perspective - View Canvas Scripting** page for the full list of scripting functions available for this component.

Perspective - View Canvas Scripting

This page details the various scripting, component, and extension functions available for **Perspective's View Canvas** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onInstanceClicked

Event is fired when a view instance is clicked.

event.index

- Object Path
  - event.index
- Type
  - Number
- Description
  - The index of the view instance.

event.params

- Object Path
  - event.params

- Type
  - Dictionary
- Description
  - The position of the view instance in relation to the canvas.

event.path

- Object Path
  - event.path
- Type
  - String
- Description
  - The path of the view instance.

event.position

- Object Path
  - event.position
- Type
  - JSON Object
- Description
  - A JSON Object representing the current position values.

event.position.top

- Object Path
  - event.position.top
- Type
  - Number

- Description
    - The top position of the view instance.

event.position.left

- Object Path
    - event.position.left
- Type
    - Number
- Description
    - The left position of the view instance.

event.position.bottom

- Object Path
    - event.position.bottom
- Type
    - Number
- Description
    - The bottom position of the view instance.

event.position.right

- Object Path
    - event.position.right
- Type
    - Number
- Description
    - The right position of the view instance.

event.size

- Object Path
  - event.size
- Type
  - JSON Object
- Description
  - A JSON Object representing the current size.

event.size.width

- Object Path
  - event.size.width
- Type
  - Number
- Description
  - The width of the view instance.

event.size.height

- Object Path
  - event.size.height
- Type
  - Number
- Description
  - The height of the view instance.

Component Functions

This component does not have component functions associated with it.

Extension Functions

This component does not have extension functions associated with it.

Perspective - Input Palette



Perspective provides a host of Input components that allow users to enter or select data, and even control a device.

Here is a complete list of Input components, and a link pointing to a page containing the component's description, properties, and an example of how to configure it.

Barcode Scanner Input

The Barcode Scanner Input component awaits for input from a barcode scanner.

Button

The Button component is used to initiate some sort of action in response to being pressed.

Checkbox

A Checkbox is a familiar component that represents a bit - it is either on (selected) or off (not selected).

DateTime Input

A DateTime Input is an easy way to select a date from a popup calendar.

DateTime Picker

The DateTime Picker component uses the calendar to select the date and time.

Dropdown

The Dropdown component is a great way to display a list of choices in a limited amount of space.

File Upload

Form

Multi-State Button

The Multi-State Button is really a series of two or more buttons, arranged in a column or row.

Numeric Entry Field

The Numeric Entry Field is similar to the standard Text Field, except that it is specialized for use with numbers.

One-Shot Button

The One-Shot Button is designed to send off a write request, and wait for a response, disabling the button until s…

Password Field

The Password Field component is similar to a Text Field component.

Radio Group

The Radio Group allows you to create multiple radio buttons in a single container.

Signature Pad

Slider

The Slider component lets the user drag an indicator along a scale to choose a value in a range.

Text Area

The Text Area component is suitable for multi-line text display and editing.

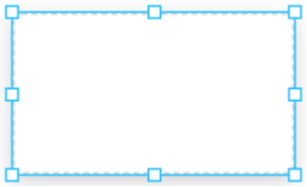
Text Field

The Text Field component is used for input of any single-line text.

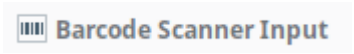
Toggle Switch

The Toggle Switch represents a bit: on (selected) or off (not selected).

Perspective - Barcode Scanner Input



Component Palette Icon:



Description

The Barcode Scanner Input component awaits for input from a barcode scanner. The component was designed for keyboard wedge scanners, as the component provides dedicated prefix and suffix properties to define scanner input. As such, it can be useful to think of the Barcode Scanner Input component as a specialized text field that does not require focus, and uses characters to decide when to accept and reject text input.

The scanner component is continuously listening, waiting for either the prefix and suffix characters to be entered, or the regex pattern to find a match. Once triggered, the component will load the scanned barcode string (excluding the prefix and suffix) into the data property for processing. The regex property can be used to extract specific fields from a scan, or validate data from the scan.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
prefix	String value that mark the start of the barcode scan capture. If a value is provided to this property, then the regex property will be ignored.	value: string
suffix	String value that marks the end of the barcode scan capture. If a value is provided to this property, then the regex property will be ignored.	value: string
regex	Regex describing the format of scans. The first capture will be used as barcode. When used, this property will pull out the first group of any regex provided. Note that this setting uses Javascript regex, as opposed to other flavors of regex. If either a prefix or suffix value is specified with a non-empty string, then this property will be ignored.	object
window	Length of buffer to monitor for regex match.	value: numeric
captureMode	Indicates which key event the component will listen for to start the barcode scan capture. Values include `keypress`, `keyup`, and `keydown`. Default value is `keypress`.	value: string
data	Barcode scans returned from scanner.	array
dataStyle	Sets a style for data returned to this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. Styles can can be set on the component before a value is scanned.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Perspective - Button



Component Palette Icon:



Description

The Button component is a versatile component, that is used to initiate some sort of action in response to being pressed. It can be used for showing status, as well. For example, you can configure buttons to be active or inactive, change color, text or any other property, and you can alter these configurations in response to conditions in your project. Button components support icons as well. For an example, see Example 2 below.

To get buttons to do things, you configure one or more Actions that occur following an Event. For instance, you might call a **script action** on the onActionPerformed component event, which triggers when the button is pressed.

The Button component has two pre-configured **variants**: Primary and Secondary.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
text	Text to display on a button. Text can also be entered directly on the button by deep selecting the component, which enables inline editing. Changes are immediately reflected in the text property field.	value: string
textStyle	Style properties that are directly applied to the text within the Button component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
primary	Toggle between default primary and secondary button style. Default is true.	value: boolean
enabled	Enables button interaction. Default is true.	value: boolean
image	An optional image embedded in the button. • **source**: The image source URL. It could be a URL to an image on the internet or Gateway, or even an embedded image. Value is string. For an image already in the Image Management console, use /system/images/`{path to your image}`. For example: /system/images/Builtin/icons/24/lightbulb_on.png • **icon**: An image path used to augment the writingState of the component by placing an image next to it. ◦ **path**: Shorthand path to icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. ◦ **color**: Color of the icon. Can also assign color in "fill" of the style property. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. • **width**: Width of the button image in pixels. Value is numeric.	object

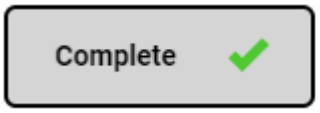
Name	Description	Property Type
	• **height**: Height of the button image in pixels. Value is numeric. • **position**: Horizontal position of the image within the button relative to the text: left, center, right, top, or bottom. Value is string. • **style**: Sets a style for the image. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
align	Aligns the text and image (if present) along the cross axis. Vertical if imagePosition is top or bottom, otherwise it's horizontal. Options are start, center, end, and stretch. Default is center.	value: string
justify	Justifies the text and image (if present) along the main axis. Horizontal if the imagePosition is top or bottom, otherwise it's vertical. Options are start, center, end, space-around, space-between, and space-evenly. Default is center.	value: string
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1



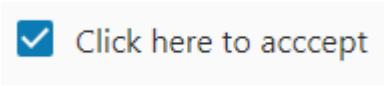
Property	Value
props.text	Complete
props.image.source	/system/images/Builtin/icons/48/check2.png
props.image.position	right
props.textStyle.color	#000000
props.justify	space-evenly
props.style.backgroundColor	#D5D5D5
props.style.borderStyle	solid
props.style.borderColor	#000000
props.style.borderWidth	2px

Example 2



Property	Value
props.text	Save
props.textStyle.color	#00AC00
props.image.source	/system/images/Builtin/icons/48/disk_green.png
props.image.position	top
props.image.width	40
props.image.height	40
props.align	end
props.justify	space-between
props.style.backgroundColor	#FFFFFF
props.style.borderStyle	inset
props.style.borderWidth	5px
props.style.borderColor	#00AC00

Perspective - Checkbox



Component Palette Icon:



Description

A Checkbox is a familiar component that represents a bit - it is either on (selected) or off (not selected). In addition, the 'triState' property can be enabled, adding a third state to represent an indeterminate value. It is functionally equivalent to the Toggle Switch component.

The DateTime Picker component has two pre-configured **variants**: Text Right and Text left.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
selected	Output value for checkbox.	value: boolean
text	Label for the checkbox. Label text can also be entered directly by deep selecting the checkbox component, which enables inline editing. Changes are immediately reflected in the text property field.	value: string
textPosition	Where to place the label text in relation to the checkbox: top, right, bottom, or left.	value: string
enabled	Whether the user can currently interact with the checkbox. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a **script action** to the onClick event, the script will fire when the user clicks on the Checkbox.	value: boolean
triState	Whether the checkbox supports a third state of "indeterminate" - effectively 'null' or 'no choice'.	value: boolean
checkedIcon	Settings for the appearance of the check box's icon when it is selected (checked). • **path**: Path to the icon source, in format: library/IconName. For more information on icons, see the **Images, and Icons in Perspective** page. Value is string. • **color**: Settings for the fill color for the icon. ◦ **enabled**: Color of the icon when enabled. Can be a named color. Value is string. ◦ **disabled**: Color of the icon when disabled. Can be a named color. Value is string. • **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Name	Description	Property Type
uncheckedIcon	Settings for the appearance of the check box's icon when it is not selected. • **path**: Path to the icon source, in format: library/IconName. For more information on icons, see the **Images, and Icons in Perspective** page. Value is string. • **color**: Settings for the fill color for the icon. ◦ **enabled**: Color of the icon when enabled. Can be a named color. Value is string. ◦ **disabled**: Color of the icon when disabled. Can be a named color. Value is string. • **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
indeterminateIcon	Settings for the appearance of the check box's icon when it is in the indeterminate state. Only applies if props.triState is set to true. • **path**: Path to the icon source, in format: library/IconName. For more information on icons, see the **Images, and Icons in Perspective** page. Value is string. • **color**: Settings for the fill color for the icon. ◦ **enabled**: Color of the icon when enabled. Can be a named color. Value is string. ◦ **disabled**: Color of the icon when disabled. Can be a named color. Value is string. • **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1

Checkbox 1 

Property	Value
props.text	Checkbox 1
props.textPosition	left
props.triState	true

Example 2

Checkbox 2 

Property	Value
props.text	Checkbox 2
props.textPosition	left
props.selected	null

Example 3

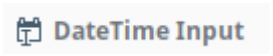


Property	Value
props.text	Checkbox 3
props.textPosition	top
props.enabled	false
props.selected	false

Perspective - DateTime Input



Component Palette Icon:



Description

A DateTime Input is an easy way to select a date from a popup calendar. Similar to the DateTime Picker component, it takes up much less real estate on the screen. Configure the date and time format in the Property Editor using the 'formattedValue' property.

The DateTime Input component has two pre-configured **variants**:

- Date and Time - Opens a calendar from which users can select a date and time.
- Time - Enables users to set a time using the up and down arrows on the component.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	Current date/time as a Date object or timestamp in milliseconds.	value: string
formattedValue	Date and time in configured format.	value: string
inputProps	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous.	object
modalStyle	Style applied to the Date picker modal (popup). Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
pickerType	Whether to display and enable picker for date only, time only, or both date and time.	value: string
minDate	Minimum date/time as a Date object or timestamp in milliseconds. If null, the minimum date is 10 years in the past from today.	value: string
maxDate	Maximum date/time as a Date object or timestamp in milliseconds. If null, the maximum date is 10 years in the future from today.	value: string
format	Template for formatting date display - must be valid moment.js format, e.g., 'MM/DD/YYYY h:mm a'.	value: string
enabled	'False' will disable any interaction with the calendar. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a **script action** to a System Event, such as an onStartup event, the script will fire when the page is loaded. Events that require user interaction, such as onClick events, will not fire with the exception of Pointer Events.	value: boolean

Name	Description	Property Type
placeholder	Text for input field to display when no date/time is selected.	value: boolean
locale	Code for localization of language and formatting. Use the dropdown to select language.	value: string
dismissOnSelect	Determines if the date picker should be dismissed when a date is selected.	value: boolean
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

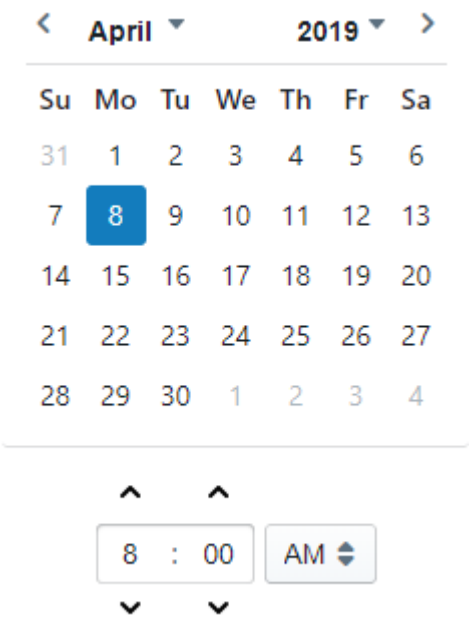
The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

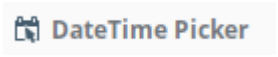


Property	Value	Style Category
props.pickerType	date	N/A
props.format	MM/DD/YYYY	N/A
props.style.borderStyle	solid	border
props.style.borderColor	#00AC00	border
props.style.borderWidth	2px	border

Perspective - DateTime Picker



Component Palette Icon:



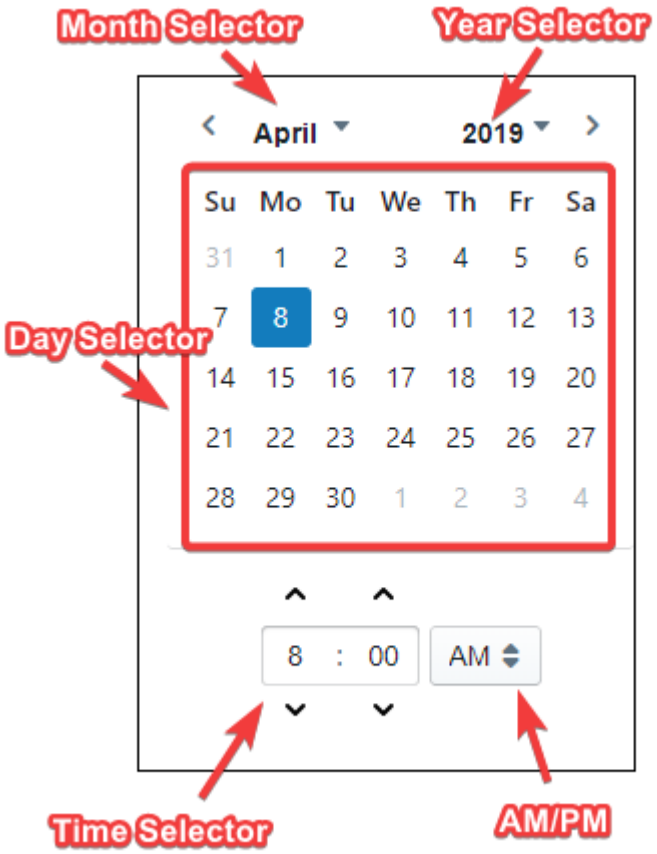
Description

The DateTime Picker component uses the calendar to select the date and time. You can choose the "pickerType" to set both date and time, or just date. Configure the date and time format in the Property Editor using the formattedValue property. To use the DateTime Picker, select the month, date, and time on the component.

The DateTime Picker component has two pre-configured **variants**:

- Date and Time - Enables users can select a date and time on a calendar.
- Date - Enables users can select a date on a calendar.

User Interface



Interaction	Description
AM/PM	Allows users to toggle between AM or PM.
Day Selector	Allows users to choose a specific day.
Month Selector	Allows users to choose a specific month.
Time Selector	Allows users to choose a specific time.
Year Selector	Allows users to choose a specific year.

Properties

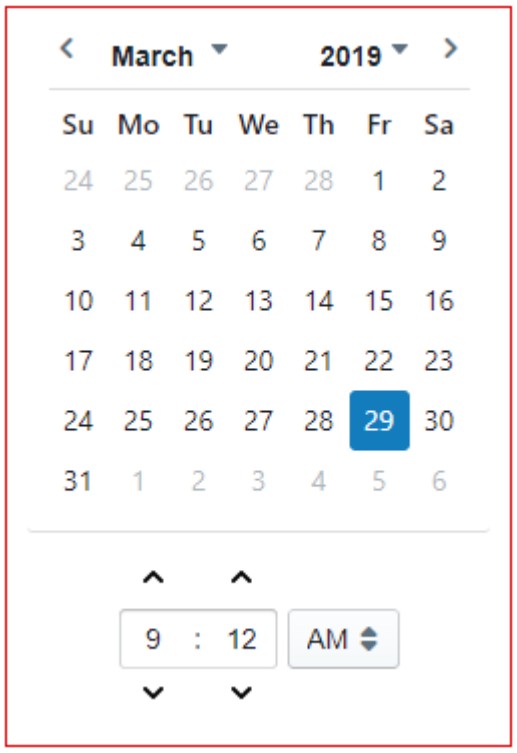
Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	Current date/time as a Date object or timestamp in milliseconds.	value: dropdown
formattedValue	Date and time in configured format.	value: string
pickerType	Whether to display and enable picker for date only or for both date and time.	value: string
minDate	Minimum date/time as a Date object or timestamp in milliseconds. If null, the minimum date is 10 years in the past from today.	value: string
maxDate	Maximum date/time as a Date object or timestamp in milliseconds. If null, the minimum date is 10 years in the future from today.	value: string
format	Template for formatting date display - must be valid moment.js format, e.g., 'MM/DD/YYYY h:mm a'.	value: string
locale	Code for localization of language and formatting. Use the dropdown to select language.	value: string
enabled	'False' will disable any interaction with the calendar. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the DateTime Picker.	value: boolean
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

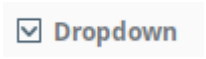


Property	Value	Style Category
props.format	MM/DD/YY hh:mm:ss a	N/A
props.style.borderStyle	solid	border
props.style.borderWidth	1px	border
props.style.borderColor	#D90000	border

Perspective - Dropdown



Component Palette Icon:



Description

The Dropdown component is a great way to display a list of choices in a limited amount of space. The current selection is shown, and the choices are presented when the user clicks on the dropdown button. There is also the capability to search for an element by typing the name of that element in the dropdown field. If the element is present, it will appear as you are typing and you can select it. If the element doesn't exist, you can define text to display that the text is not found by configuring the **noResultsText** property.

The choices that are displayed in the Dropdown depend on what Elements are defined in the **options** property of the Property Editor. The **placeholder** property defines what text is shown before any of the choices are selected. For example, you can have the word 'Select...' to inform the user to select any of the Elements from the dropdown list.

There is also a **multiSelect** property which allows the user to select multiple elements from the dropdown. Selected elements can be deleted by clicking the 'x' icon.

The Dropdown component has two pre-configured **variants**:

- Single Selection: Default layout that displays a list of choices of which the user can select one.
- Multi-Selection: Layout with a list of choices of which the user can select more than one.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	The result of current selections (input) after any processing.	variable, based on which item in props.options property is selected.
options	An array of objects for each dropdown option. • **value**: Actual value to be matched by the input or selection. The type of this property is initially a Value-type, but it can be converted to an Object-type or Array-type. Doing so will populate the PROPS.value property with the entire object/array, allowing a single selection on the dropdown to return multiple values. Property type is variable. **Note**: Each option must have a unique value. • **label**: Text to display in the menu representing this option. Value is string. • **isDisabled**: Whether this option is currently disabled from selection. If set to true, option will not be selectable, and will use a grey font assuming another text color isn't being applied. Value is boolean.	array
multiSelect	Enable multiple selected values. Default is false.	value: boolean
placeholder	Settings for the text displayed when value is empty. Click to see the **placeholder properties**.	object
enabled	If set to false, component is disabled. Field will not focus and dropdown will not be interactive. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a	value: boolean

Name	Description	Property Type
	script action to the **onClick** event, the script will fire when the user clicks on the Dropdown.	
search	Enter text to start search. • **enabled**: Whether options are searchable by typing text into the field. Default is true. Value is boolean. • **matching**: Whether search string must match from the start or may match any position of an option: start or any. Value is string. • **noResultsText**: Text to display in dropdown when no option matches the search. Default is "No results found." Value is string. • **searchParam**: The text being searched for. Value is string.	object
showClearIcon	Whether to display a button that the user can use to clear the selection. Default is false.	value: boolean
allowCustomOptions	Whether a user may enter a custom value to be submitted. Default is false. While set to True, typing a value that doesn't match one of the existing options from the session will provide the user with a "Create" option. Selecting the Create option will set `props.value` to the new option. Creating a custom option in this way does not change the value of `props.options`. The custom option is simply a means to allow the user to type a custom value into the component.	value: boolean
textAlign	Aligns the value(s) and/or placeholder text displayed within the dropdown. Valid values are left, center, and right. textAlign within the dropdown modal itself may be overridden or set separately using props.dropdownOptionStyle.textAlign.	value: string

Name	Description	Property Type
minMenuHeight	Minimum height of the dropdown menu. If the minimum height is not available under the component, and space is available above the component, the dropdown will flip to display options above the component. Default value is 150.	value: numeric
maxMenuHeight	Maximum height of the dropdown menu before it becomes scrollable. Default value is 350.	value: numeric
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
dropdownOptionStyle	Sets a style for the dropdown options. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

placeholder Properties

Name	Description	Property Type
text	Prompt text to display when no options are selected.	value: string
color	Color of placeholder text. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See Color Selector.	color
icon	Settings for an icon used as a placeholder. • **path**: Shorthand path to the icon source. Format is library/iconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. • **color**: Color of the placeholder icon, if it exists. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. • **style**: Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. Three additional available settings are as follows: ◦ **margin**: Margin around the icon, in pixels. Default is 0px 8px -3px 0px. Value is numeric. ◦ **width**: Width of the icon in pixels. Default is 16px. Value is numeric. ◦ **height**: Height of the icon in pixels. Default is 16px. Value is numeric.	object

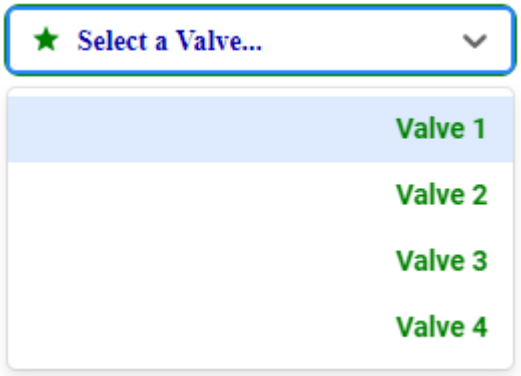
Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective

component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1



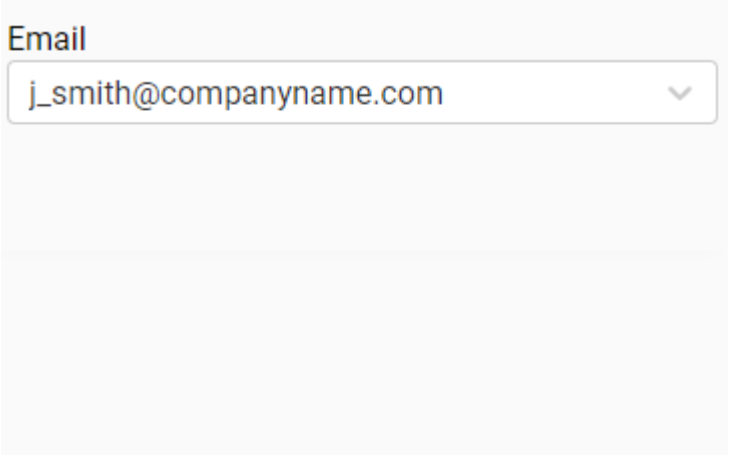
Property	Value
props.options.0.value	`[default]_Dairy_/Bldg25/valve1`
props.options.0.label	`Valve 1`
props.options.1.value	`[default]_Dairy_/Bldg25/valve2`
props.options.1.label	`Valve 2`
props.options.2.value	`[default]_Dairy_/Bldg25/valve3`
props.options.2.label	`Valve 3`
props.options.3.value	`[default]_Dairy_/Bldg25/valve4`
props.options.3.label	`Valve 4`
props.placeholder.text	`Select a Valve...`
props.placeholder.color	`#0000AC`
props.placeholder.icon.path	`material/grade`
props.placeholder.icon.color	`#008000`
props.style.borderStyle	`solid`
props.style.color	`#0000D9`
props.style.fontFamily	`garamound`
props.style.fontSize	`16px`
props.style.fontWeight	`bold`
props.style.borderWidth	`2px`
props.style.borderColor	`#008000`
props.style.borderRadius	`8px`

Property	Value
props.dropdownOptionStyle.color	`#008000`
props.dropdownOptionStyle.fontSize	`14px`
props.dropdownOptionStyle.fontWeight	`bold`
props.dropdownOptionStyle.textAlign	`right`

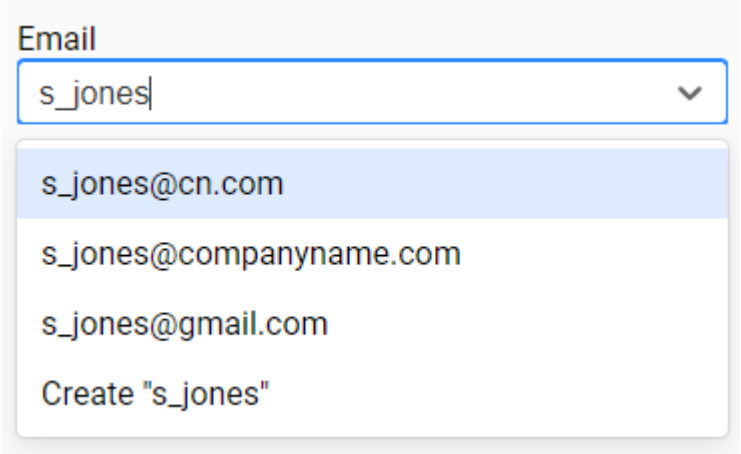
Example 2

In this example, we have a dropdown list with an expression binding on the options property. There is also a label on the view with the word "Email" as its text.

A default email address of **j_smith@companyname.com** is set as the starting value for the component.



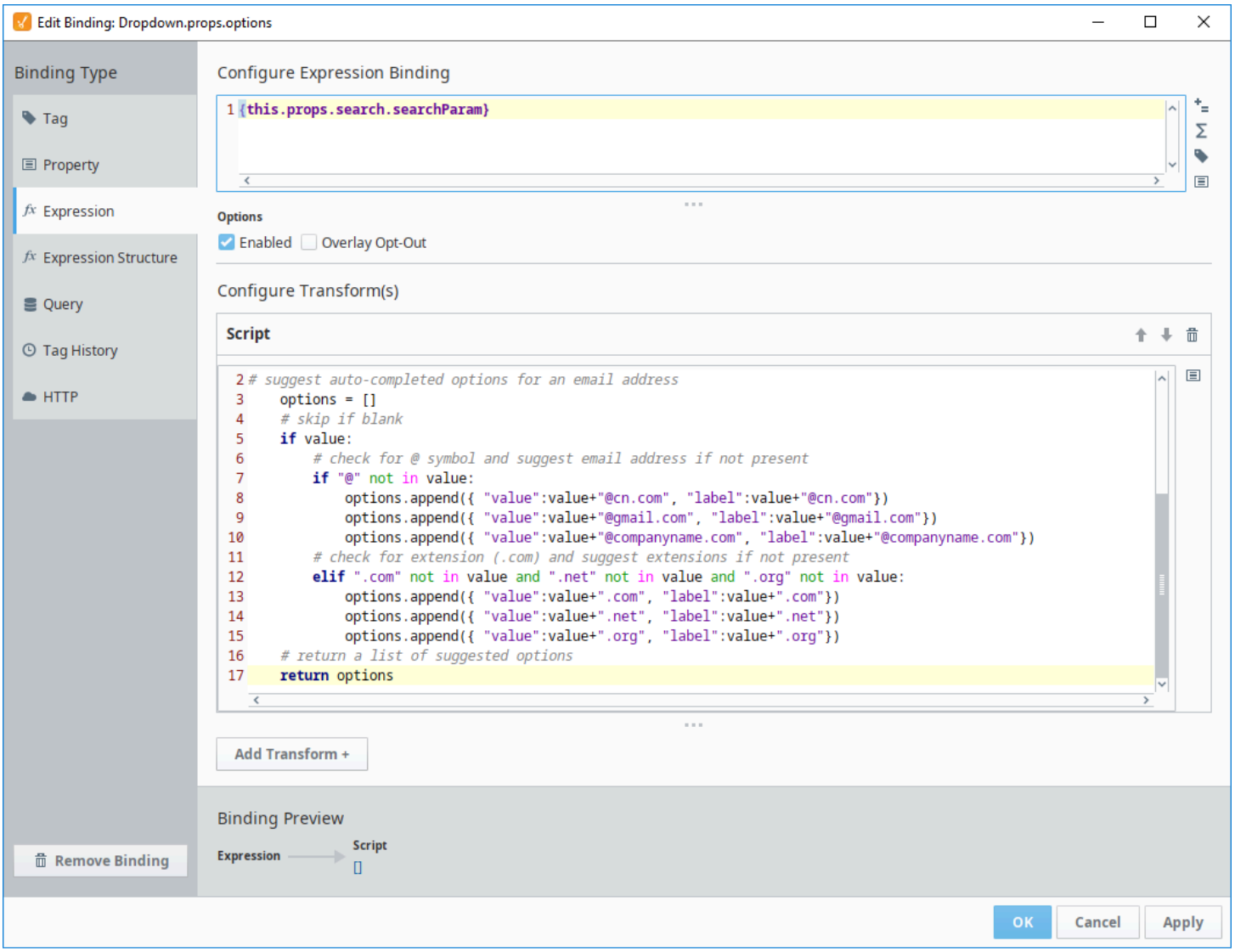
As the user starts entering characters for an email address, the dropdown list provides typeahead options of the entered text plus three possible email options, @'cn.com', '@companyname.com', or '@gmail.com'.



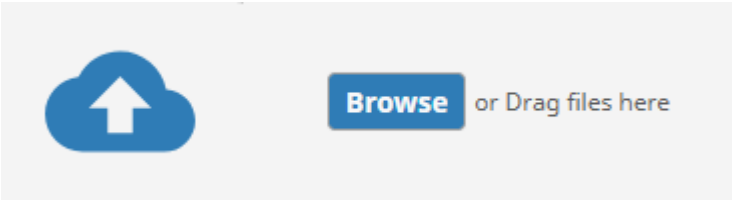
Property	Value
value	`j_smith@companyname.com`
options	(Bound to an expression. See example below.)
props.search.enabled	`true`

transform code

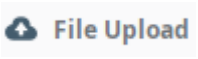
```
# suggest auto-completed options for an email address
options = []
# skip if blank
if value:
    # check for @ symbol and suggest email address if not present
    if "@" not in value:
        options.append({ "value":value+"@cn.com", "label":value+"@cn.com"})
        options.append({ "value":value+"@gmail.com", "label":value+"@gmail.com"})
        options.append({ "value":value+"@companyname.com",
"label":value+"@companyname.com"})
    # check for extension (.com) and suggest extensions if not present
    elif ".com" not in value and ".net" not in value and ".org" not in value:
        options.append({ "value":value+".com", "label":value+".com"})
        options.append({ "value":value+".net", "label":value+".net"})
        options.append({ "value":value+".org", "label":value+".org"})
# return a list of suggested options
return options
```


Edit Binding: Dropdown.props.options
Binding Type
Tag
Property
Expression
Expression Structure
Query
Tag History
HTTP
Remove Binding
Configure Expression Binding
1 {this.props.search.searchParam}
Options
Enabled
Overlay Opt-Out
Configure Transform(s)
Script
2 # suggest auto-completed options for an email address
3 options = []
4 # skip if blank
5 if value:
6 # check for @ symbol and suggest email address if not present
7 if "@" not in value:
8 options.append({ "value":value+"@cn.com", "label":value+"@cn.com"})
9 options.append({ "value":value+"@gmail.com", "label":value+"@gmail.com"})
10 options.append({ "value":value+"@companyname.com", "label":value+"@companyname.com"})
11 # check for extension (.com) and suggest extensions if not present
12 elif ".com" not in value and ".net" not in value and ".org" not in value:
13 options.append({ "value":value+".com", "label":value+".com"})
14 options.append({ "value":value+".net", "label":value+".net"})
15 options.append({ "value":value+".org", "label":value+".org"})
16 # return a list of suggested options
17 return options
Add Transform +
Binding Preview
Expression
Script
[]
OK
Cancel
Apply

Perspective - File Upload



Component Palette Icon:



Description

The File Upload component allows users to upload files to the Gateway or other locations from a Perspective session using a script action on the onFileReceived component event. For an example, see **Download and Upload Files** page.

The component has different appearances based on its width. When initially dragged onto your View, a Browse button will be visible and paired with "Drag files here" text. At smaller widths, the component defaults to a simple cloud icon and no text. This icon can be changed using the fileUploadIcon property settings.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
enabled	**New in 8.3.5** Determines whether or not the component is enabled. Note that the color and style enabled/disabled properties are dependent on this property state. Default is true.	value: boolean
maxUploads	The maximum number of concurrent (simultaneous) uploads to allow. Default is 5.	value: integer
supportedFileTypes	An array of string values, indicating what file types are allowed to be uploaded. Example values are `pdf` or `txt`.	array
fileSizeLimit	Specifies the maximum size of each uploaded file, in megabytes (MB). Default is 10 MB. Note that values over 20 MB are not allowed, unless the max-file-size property in the **web.xml** configuration file is set to a higher limit, as the max-file-size property overrides this setting.	value: integer
fileUploadIcon	Determines the icon used when the File Upload component is small. • **path**: Shorthand path to the icon source. Format is library/iconName. The **materials icon library** is a the primary source for icons. Value is string. • **color**: Color of the placeholder icon, if it exists. Can be chosen from a color wheel, a color palette, or entered as an RGB or HSL value. See **Color Selector** for more information. ◦ **enabled**: **New in 8.3.5** Color of the icon when the parent enabled property is set to true. ◦ **disabled**: **New in 8.3.5** Color of the icon when the parent enabled property is set to false. • **style**: Sets a style for the icon.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Name	Description	Property Type
	• **enabled**: New in 8.3.5 Style of the component when the parent enabled property is set to true. • **disabled**: New in 8.3.5 Style of the component when the parent enabled property is set to false.	

Scripting

See the **Perspective - File Upload Scripting** page for the full list of scripting functions available for this component.

Download and Upload Files

Downloading and uploading files from a Perspective session typically involves storing and retrieving files from a database. A table will store all of the available files, and each row of the table represents a new file. This allows for long term storage that is accessible from any project.

Query Examples

The examples on this page show suggested methods of uploading files from a session, as well as how to download them, using the **File Upload component**. Before following along with the examples on this page, you'll need to create a table in the database that will hold the files. This process can vary by database, along with the column datatypes.

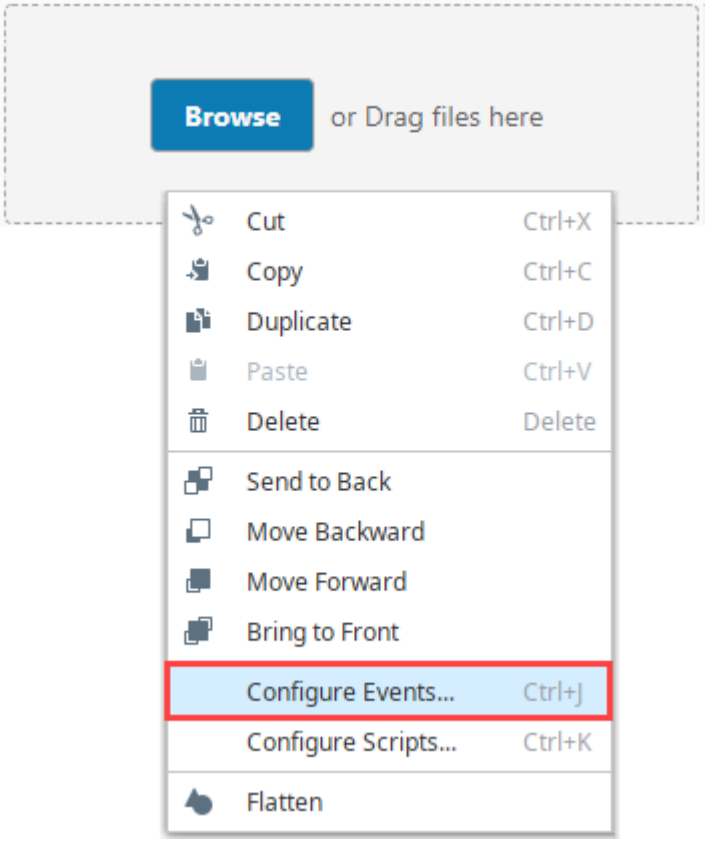
For the sake of brevity, the example assumes the files will be stored and retrieved from a SQL Server database. You may need to modify the query examples on this page if using a different database. The **files** database table used by these examples contains the following columns:

- **id** - integer, primary key, identity
- **filename** - varchar (255)
- **filedata** - varbinary (MAX)

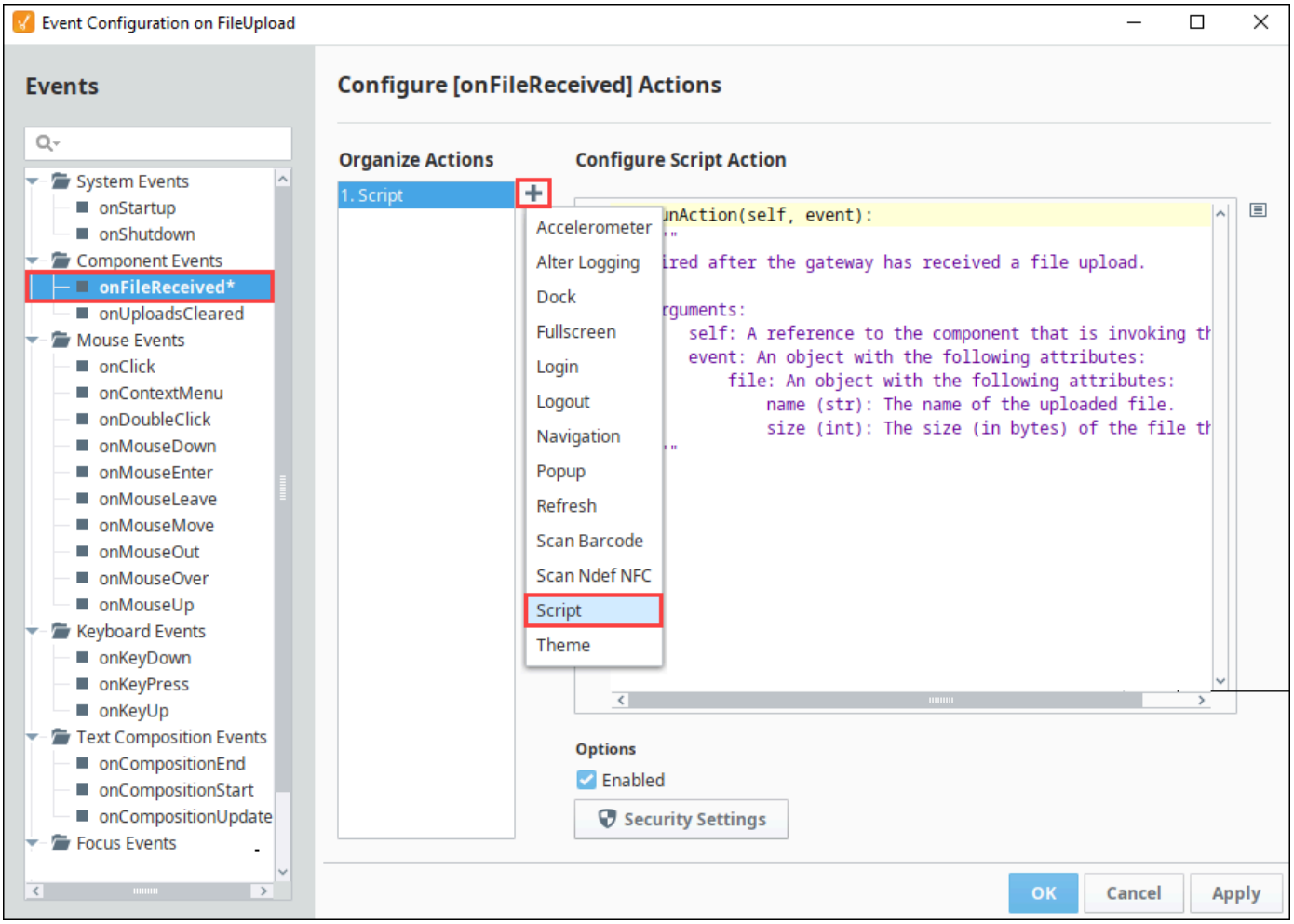
Uploading a File

1. Add the File Upload component to a view. The File Upload component has everything we need to upload a file into the database.

2. Right-click on the File Upload component and select **Configure Events**.



3. Select the **onFileReceived** event and click the **Add** + icon to add a **script** action to it.



4. Add the following script to the script action:

```
# Grab the file name and data
filename = event.file.name
filedata = event.file.getBytes()

# Use a query to insert the file
query = "INSERT INTO files (filename, filedata) VALUES(?,
CONVERT(varbinary(MAX), ?))"
args = [filename, filedata]
db = "myDatabase"
system.db.runPrepUpdate(query, args, db)
```

> **NOTE**
>
> As mentioned above, the query will vary based on the database used.

5. Click **OK**.

You can test out the upload functionality by dragging a file onto the File Upload component, either from a session or the Designer while it's in preview mode. After the file is uploaded, a successful upload message will appear and your file will be present in the created database table.

Downloading a File

To download a file that is stored in the database in Perspective, we will want to use the **system.perspective.download** function. This will allow us to download the file data that we receive from the database.

This example will show you how to do several things:

- Create a **named query**, that will return the contents of our file database table.
- Create a table component, that shows a listing of potential files to download, using the Named Query above in conjunction with a Named Query binding.
- Add a Button component that will allow users to download a file, assuming one of the rows in the table component is selected.

To download a file:

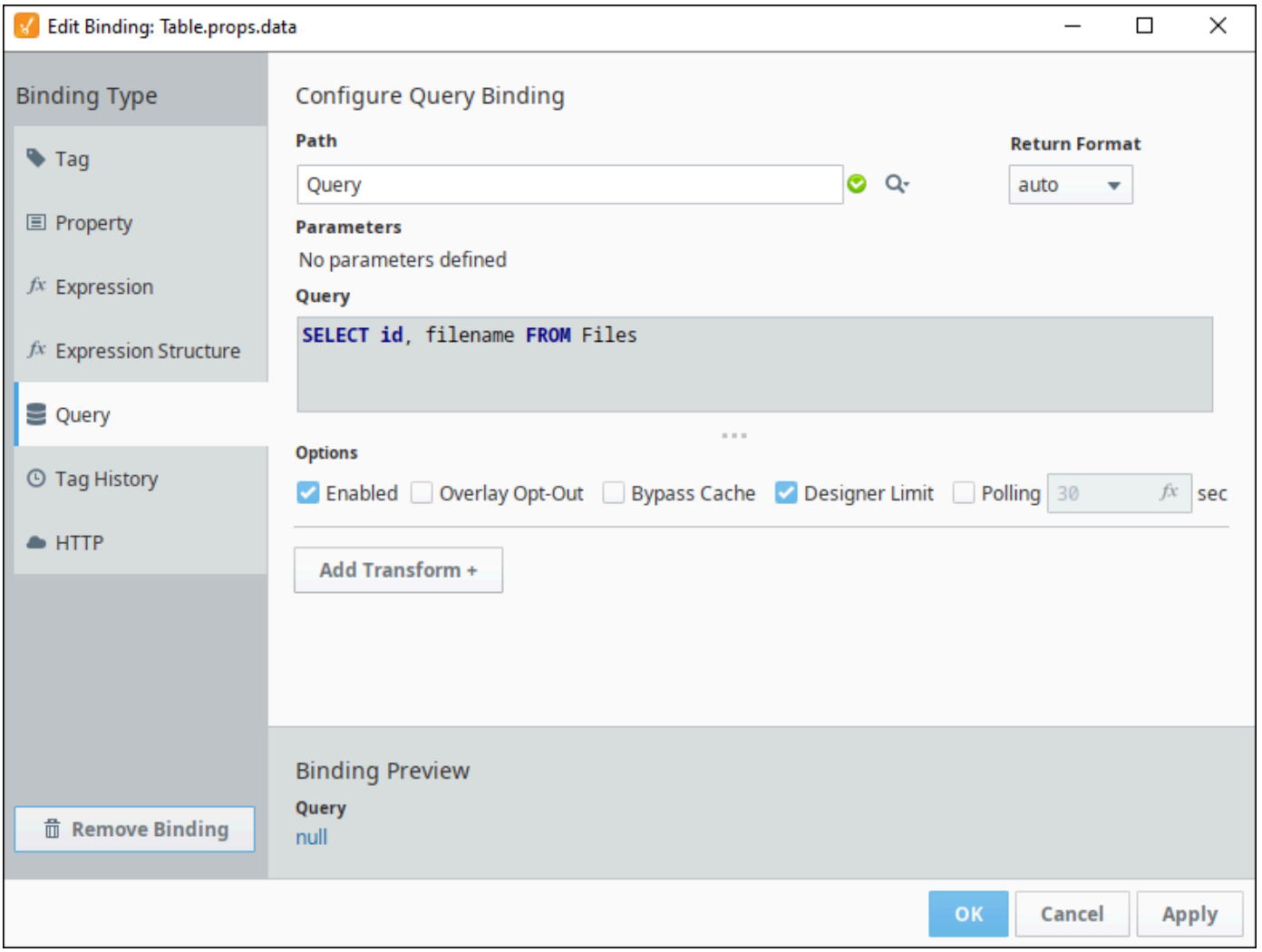
1. Create a Named Query that we will use to pull a list of files out of the database table. We're using a Named Query here since a Named Query binding is the easiest way to run a query when the view loads.

The query should pull out the id of the row which we can use to later query the data, as well as the filename which the user can use to identify the file.

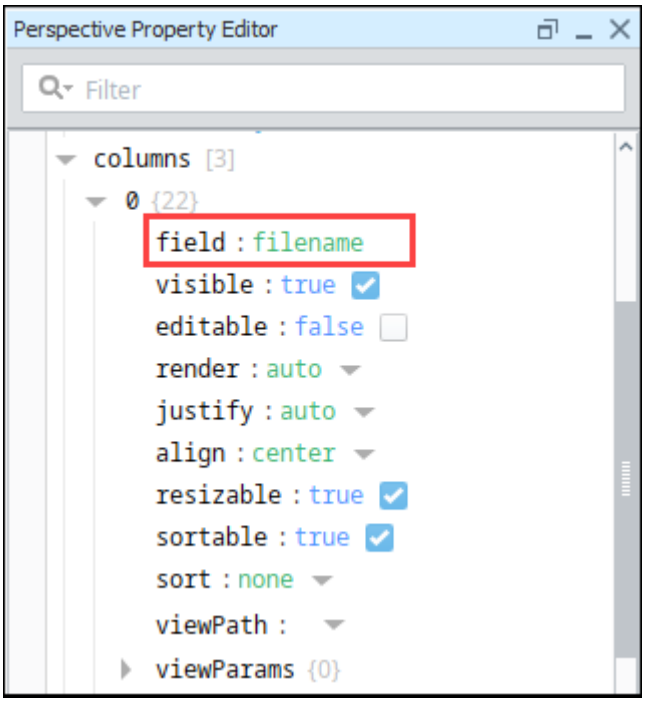
```
SELECT id, filename FROM Files
```

2. On a view, add a Table component. This will display a list of all files we currently have in the database table.

3. Set up a binding on the Table's data property. The binding should be a Query type, and it should use the query that we just made. We want to return the data in a JSON format, and you can enable polling so that it automatically updates if new files get uploaded.



4. On the Table's columns property, add an array element. Set **columns.0.field** to the name of the column that holds the filename. This will display only the filename column, as the id column does not need to be visible.

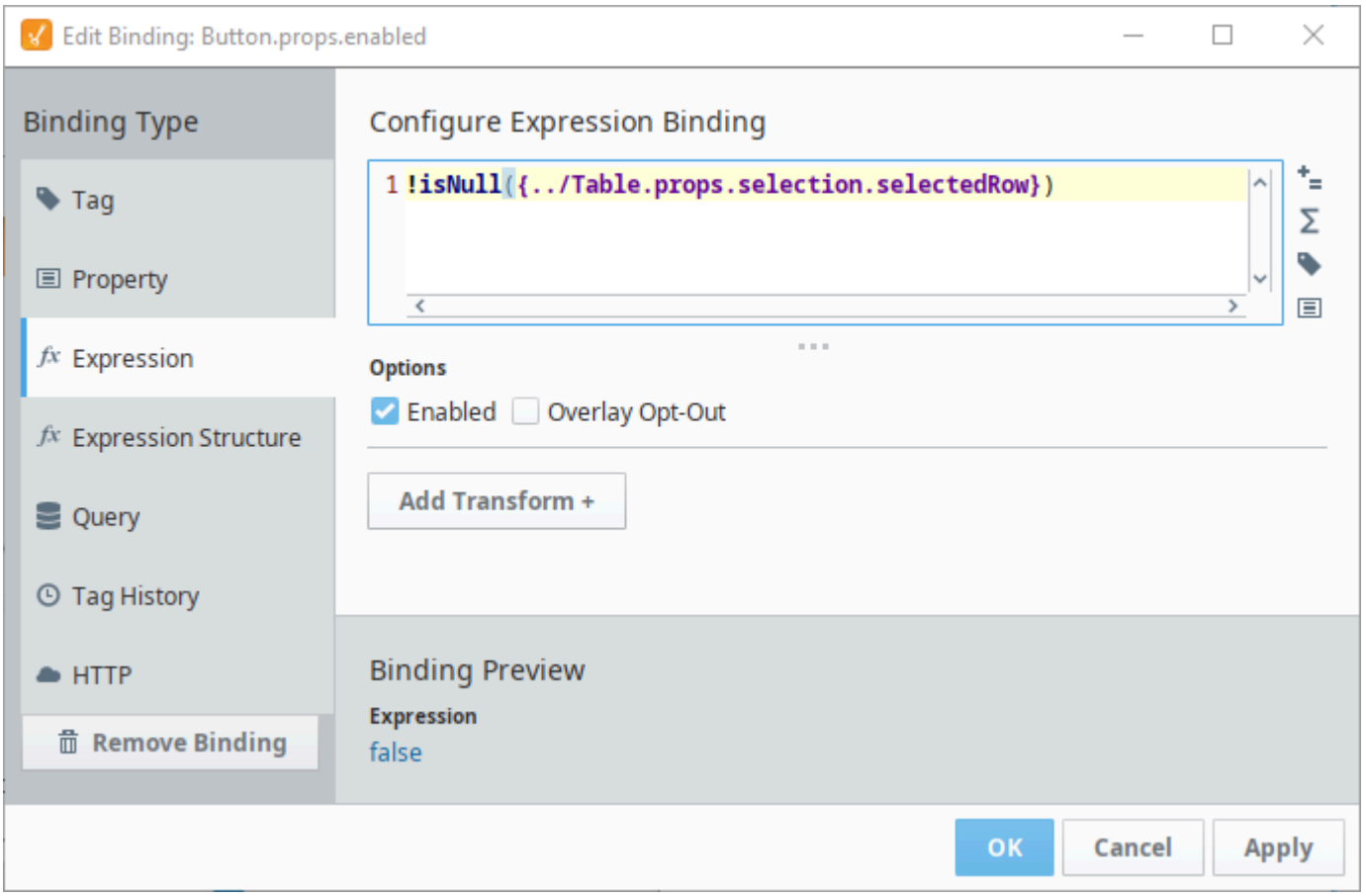


5. Add a Button to the view. This button will be used to download the file after the user has made a selection. However, we also want to make sure the user can't press the button unless a row in the table is selected.

6. On the Button's enabled property, configure a binding. The binding type should be an expression. The expression should check to see if the Table's selected row is null, and invert it.

```
!isNull({../Table.props.selection.selectedRow})
```

   This will disable the component if no row is selected, to prevent the user from trying to download without making a selection.



7. Right-click on the Button and go to **Configure Events**.

8. Select the **onActionPerformed** event, click the **Add** + icon to add a script action to it.

9. Add the following script to the script action:

```
# Grab the selected row
selectedRow = self.getSibling("Table").props.selection.selectedRow

# Use the selected row to grab the id of the file at that row
id = self.getSibling("Table").props.data[selectedRow].id

# Use the id to grab the file data out of the database, along with its
corresponding name.
query = "SELECT filename, filedata FROM Files WHERE id = ?"
args = [id]
data = system.db.runPrepQuery(query, args)

# Pull out the file name and data
filename = data[0][0]
filedata = data[0][1]

# Download the file
system.perspective.download(filename, filedata)
```

10. Test the script by selecting a row in the table and clicking on the button while in Preview mode.

Perspective - File Upload Scripting

This page details the various scripting, component, and extension functions available for Perspective's **File Upload** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onFileReceived

Provides a chance to handle file data uploaded to the component.

> **NOTE**
>
> This component event is designed to be used in tandem with a script action. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event.file.name

- Object Path
  - event.file.name
- Type
  - String
- Description
  - The name of the uploaded file.

event.file.size

- Object Path
  - event.file.size
- Type
  - Integer
- Description
  - The size of the uploaded file in bytes.

event.file.copyTo(filePath)

- Object Path
  - event.file.copyTo()
- Description
  - Saves the uploaded file at a location accessible to the Gateway.
- Parameters
  - String **filePath** - The path to where the file should be saved on the Gateway.
- Return
  - None

event.file.getBytes()

- Object Path
  - event.file.getBytes()
- Description
  - Fetches the incoming file data. Suitable for further data processing.
- Parameters
  - None
- Return

- byteArray - The raw data of the incoming file.

event.file.getString()

- Object Path
  - event.file.getString()
- Description
  - Fetches the incoming file data and attempts to parse it as a string via UTF-8 (Eight-bit UCS Transformation Format) encoding. Although this defaults to UTF-8, other character sets can be used. For example, getString("UTF_16BE").
- Parameters
  - None
- Return
  - String - The raw data of the incoming file as a string.

onUploadsCleared

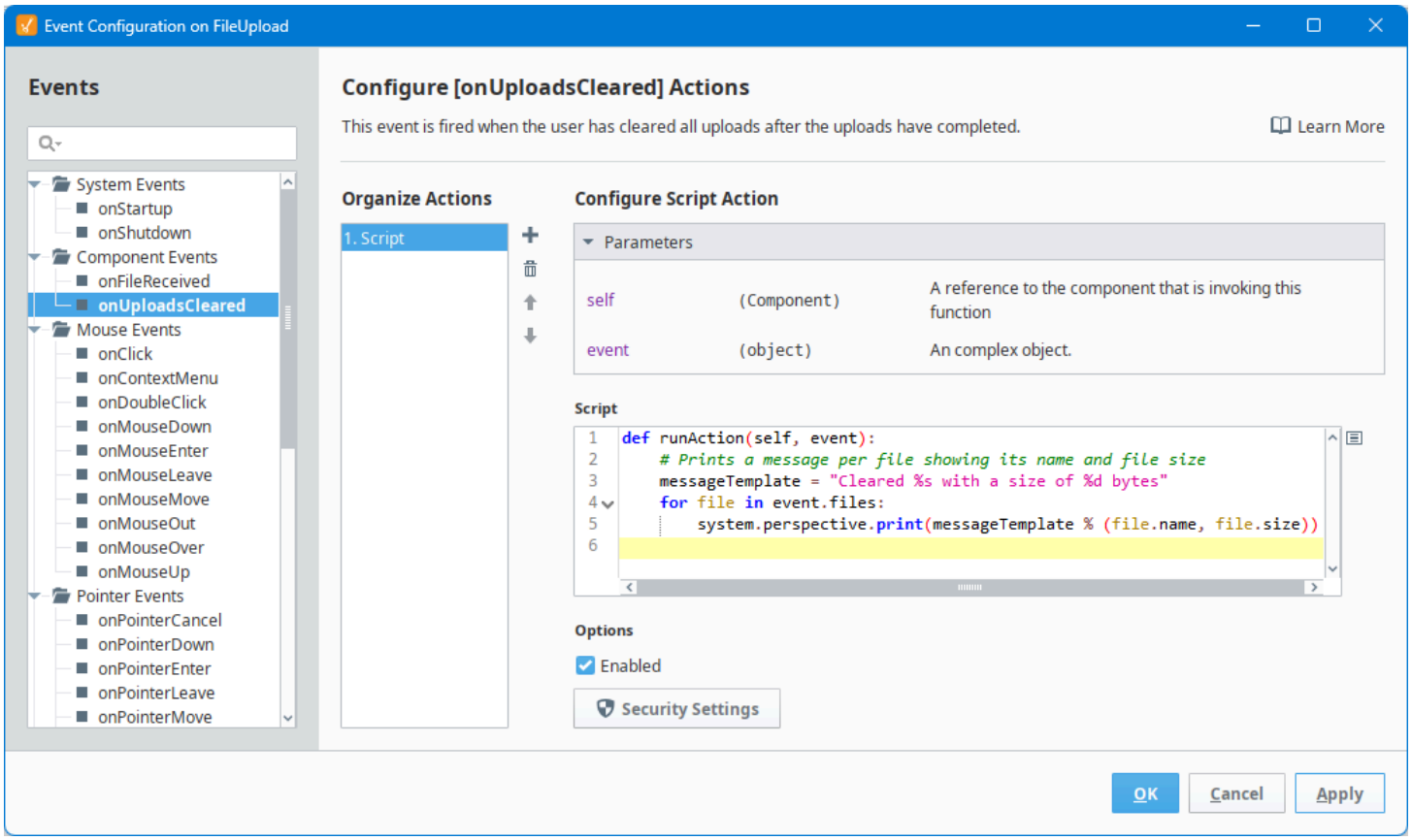
This event is fired when the user has cleared all uploads, but not while uploads are still in progress.

> **NOTE**
> This component event is designed to be used in tandem with a script action. Within the script action, special properties and methods are available on the event object, which is passed to the script action as a parameter.

event.files

- Object Path
  - event.files
- Type
  - List
- Description

- A list containing a number of PyJsonObjectAdapters, where each adapter contains a `name` (string) and `size` in bytes (integer bytes)



Event.Files Example

```
# Prints a message per file showing its name and file size
messageTemplate = "Cleared %s with a size of %d bytes"
for file in event.files:
        system.perspective.print(messageTemplate % (file.name, file.size))
```

Accessing File Name Individually

```
# Alternatively you can access individual items:
messageTemplate = "The name of the first file cleared is: %s"
system.perspective.print(messageTemplate % event.files[0].name)
```

Component Functions

.clearUploads()

- Description

- Resets the File Upload component to its default state

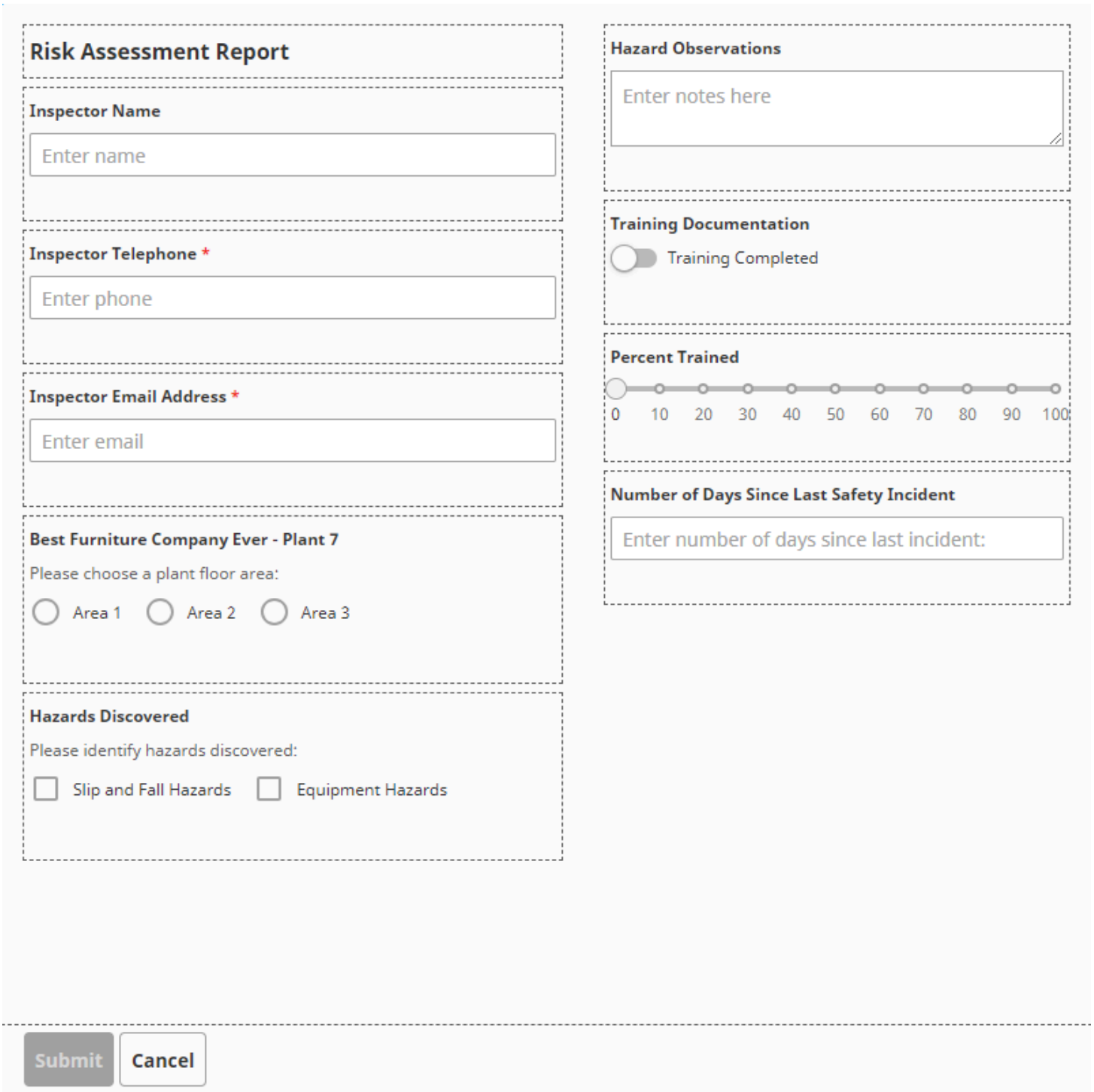
> **NOTE**
>
> clearUploads() does not remove or delete uploaded files from the Gateway. Clearing uploads does not undo any actions triggered by the onFileReceived() Component Event.

- Parameters
  - None
- Return
  - Nothing

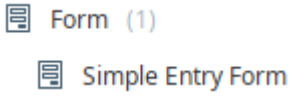
Extension Functions

This component does not have extension functions associated with it.

Perspective - Form



Component Palette Icon:



Description

The Form component is a versatile container for creating dynamic and responsive forms in Perspective. It simplifies collecting and validating user input, making it easy to create professional and interactive forms.

Key features include:

- **Responsive Design**: Automatically adapts layouts for different screen sizes.
- **Validation Support**: Provides built-in and custom validation rules.
- **Flexible Layouts**: Configure rows, columns, and widgets to meet your needs.
- **Dynamic Visibility**: Conditionally display or enable form elements based on user input.

Properties

These properties define the behavior and structure of the Form at the top level. They live directly under `props` on the component and control things like the form name, whether interaction is allowed, how many columns the form contains, and how validation information is tracked. Most properties support binding options. For more information, see **Types of Bindings in Perspective**.

Name	Description	Property Type
name	The name of the form. This name will be used by handlers to identify forms and sort form submissions.	String
disabled	Disables all inputs and actions in the form. Default is false.	Boolean
readOnly	Sets all inputs to read-only. Inputs remain focused but cannot be edited. Default is false.	Boolean
columns	Array of column configuration objects defining the form structure. See the **Columns Properties** section below for more detail on configuration options.	Array
actions	Configuration object for Submit and Cancel buttons, including text, style, and behavior settings. See the **Actions Properties** section below for more detail on configuration options.	Object
context	Provides additional context for the form, including metadata and configurations.	Object
data	Stores user input values using unique widget IDs.	Object
validation	Read-only object containing live validation results, including references to any widgets that are currently invalid.	Object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Columns Properties

The columns object controls how columns are arranged and holds the list of individual column definitions. As columns are defined, more properties become available to configure. Most notable is the items array, which is included at the columns level and again at the rows level. All possible column properties are included in the tables below based on their property path.

Name	Description	Property Type
align	Specifies how rows are aligned horizontally within this column. Inherited by all items of this column. Options include start, end, center, and stretch.	String
justify	Specifies how rows are aligned vertically within this column. Inherited by all items of this column. Options include start, end, center, space-between, and space-around.	String
items	An array of column configuration objects. The number of entries in 'items' determines how many columns the form displays. See **columns.items[x] Properties** below for more details.	Array
style	Sets a style for all columns. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

`columns.items[x]` Properties

The items array appears under `props.columns.items[x].*` and control visibility and styling for an individual column, and contain the rows that make up that column.

Name	Description	Property Type
visible	Controls whether this column is shown. Default is true.	Boolean
rows	A rows configuration object for the columns item. See **Rows Properties** below for more details.	Object
style	Sets a style for the column item. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

`rows` Properties

The rows object appears under `props.columns.items[x].rows` and controls how rows items inside a single column are laid out.

Name	Description	Property Type
align	Specifies how widgets are aligned horizontally within this row. Inherited by all items of this row. Options include start, end, center, and stretch.	String
justify	Specifies how widgets are aligned vertically within this row. Inherited by all items of this row. Options include start, end, center, space-between, and space-around.	String
items	An array of row configuration objects for this column. Each entry represents a single row. The number of entries in `items` determines how many rows the column displays. See **rows.items[x] Properties** below.	Array
style	Sets a style for the rows in the column. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

`rows.items[x]` Properties

The items array for rows appears under `props.columns.items[x].rows.items[x].*` and groups widgets together to control their alignment, visibility, and interaction conditions.

Name	Description	Property Type
align	Specifies how widgets are aligned horizontally within this row. Overrides the value inherited from the parent row configuration. Options include start, end, center, and stretch.	String
justify	Specifies how widgets are aligned vertically within this row. Overrides the value inherited from the parent row configuration. Options include start, end, center, space-between, and space-around.	String
visible	Controls whether this row is shown. Default is true.	Boolean
enabled	Controls whether the widgets in this row accept input. Default is true.	Boolean
widgets	An array of widget objects that exist within this row. See the **Widgets** section for details.	Array
style	A style object applied to the row. Full menu of **style options** is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	Object
visibleWhen	• **id**: The identifier of the widget or value source used when evaluating this condition as a string. • **index**: An optional index used when the source contains multiple values such as within arrays or lists, provided as a number. • **value**: The comparison value evaluated to determine whether the row becomes visible and may be any type.	Object
enabledWhen	• **id**: The identifier of the widget or value source used when evaluating this condition as a string. • **index**: An optional index used when the source contains multiple values such as within arrays or lists, provided as a number.	Object

Name	Description	Property Type
	• **value**: The comparison value evaluated to determine whether the row becomes enabled and may be any type.	

Actions Properties

Actions appear under `props.actions.*` and define the behavior, layout, and visibility of the action bar that contains the form's Submit and Cancel buttons. The default form layout places this action bar at the bottom, but its position can be adjusted as needed.

Name	Description	Property Type
disabled	Disables all actions in the form. When true, Submit and Cancel buttons do not accept interaction. Default is false.	Boolean
fixed	Determines whether the action bar remains fixed in place, even when scrolling.	Boolean
layout	Specifies how the contents of the action bar are arranged. Options include: row, column, row-reverse, and column-reverse	Object
align	Aligns actions along the cross axis. Actions can be aligned from the start, end, or center. The alignment layout will be vertical when set to row and horizontal when set to column.	String
justify	Justifies actions along the main axis. Actions can be justified from the start, end, center, space-between, space-around, or space-evenly. The justification layout will be horizontal when set to row and vertical when set to column.	String
submit	Configuration for the submit action. See **Submit Properties** below for more details.	Object
cancel	Configuration for the cancel action. See **Cancel Properties** below for more details.	Object
style	Sets a style for the actions. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Submit Properties

The Submit button configuration appears under `props.actions.submit.*` and defines the text, icon, behavior, event triggers, submission handling, response display, and reset logic associated with the submit action.

The Submit button is always rendered as part of the Form component's action bar. While it can be disabled using the `disabled` property, it cannot be hidden. This is intended behavior, as the Form

component is designed to provide a consistent submission mechanism and ensure that users always have access to a primary action for submitting form data.

Name	Description	Property Type
disabled	Disables the Submit button, preventing user interaction. Default is false.	Boolean
text	The text displayed on the Submit button.	String
icon	Configures the icon shown on the Submit button, including its path, color, and style. • **path**: The icon path used for the Submit button as a string. • **color**: The color applied to the icon as a string. • **style**: A style object applied to the submit icon. A full menu of **style options** is available and you can also specify a **style class**.	Object
fireComponentEvent	When true, triggers the `onSubmitActionPerformed` component event script.	Boolean
fireSubmissionEvent	When true, sends form data to a Gateway-scoped Form Submission event script and enables the `submissionHandler` property.	Boolean
submissionHandler	The name of the Form Submission handler script. Only appears when `fireSubmissionEvent` is true.	String
includeDisabledFieldValues	Includes values from disabled widgets in the submission payload when true. Default is false.	Boolean
includeHiddenFieldValues	Includes values from hidden widgets in the submission payload when true. Default is false.	Boolean
autoDisable	Automatically enables or disables the Submit button based on form validity and whether a submission is in progress.	Boolean
resetOn	Determines when the form resets after submission. Determines when the form resets after submission.	String

Name	Description	Property Type
	• **success**: Resets the form only after a successful submission response. • **action**: Resets the form immediately after the submit action occurs.	
awaitResponse	When true, the form waits for a Gateway response before applying reset behavior. Only applies when `fireSubmissionEvent` is true.	Boolean
responseDisplay	Controls how submission responses are presented, including layout and style for the title and message areas. • **layout**: Specifies how the response area is arranged in relation to the actions as a string. Supported options include: ◦ **row**: Places the response area horizontally next to the action bar. ◦ **column**: Places the response area vertically above or below the action bar. ◦ **row-reverse**: Places the response area horizontally but in reversed order. ◦ **column-reverse**: Places the response area vertically in reversed order. • **titleStyle**: A style object applied to the title text of the response. • **messageStyle**: A style object applied to the response message text. • **style**: A style object applied to the response display container. Note that a full menu of **style options** is available for each of the style properties. You can also specify a **style class**.	Object
style	A style object applied to the Submit button. Full menu of **style options** is available for text, background,	Object

Name	Description	Property Type
	margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	

Cancel Properties

Cancel button configuration appears under `props.actions.cancel.*` and define how the cancel action behaves, including button text, icon configuration, component event triggers, and whether the form resets when the button is pressed.

The Cancel button is always rendered as part of the Form component's action bar. While it can be disabled using the `disabled` property, it cannot be hidden. This is intended behavior, as the Form component is designed to provide a consistent submission mechanism and ensure that users always have access to a primary action for submitting form data.

Name	Description	Property Type
disabled	Disables the Cancel button, preventing user interaction.	Boolean
text	The text displayed on the Cancel button.	String
icon	Controls the appearance of the Cancel button's icon, including its path, color, and style. • **path**: The icon path used for the Cancel button as a string. • **color**: The color applied to the icon as a string. • **style**: A style object applied to the cancel icon. A full menu of **style options** is available and you can also specify a **style class**.	Object
style	A style object applied to the Cancel button. Full menu of **style options** is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	Object
fireComponentEvent	When true, triggers the `onCancelActionPerformed` component event script.	Boolean
reset	Resets all form fields to their default values when the Cancel button is pressed.	Boolean

Widgets

Widgets live at deeper paths under `props.columns.items[x].rows.items[x].widgets[x]`. However, they function as their own complete layer of configuration where each widget represents a single form input like a text field, dropdown, toggle, or slider. All widgets share a common set of base properties, and each widget type introduces its own additional configuration options.

Widget Properties

Name	Description	Property Type
id	A unique identifier for the widget. Used in scripts and the form's `data` object.	String
domID	The HTML ID attribute applied to the widget for styling or scripting purposes.	String
type	Determines which widget configuration is available. See the **Widget Types** section for a list of available widgets.	String
visible	Controls whether the widget is shown. Default is true.	Boolean
enabled	Controls whether the widget accepts input. Default is true.	Boolean
readOnly	Sets all inputs to read-only. Inputs remain focused but cannot be edited. This property becomes available when text, email, URL, password, number, or tel widgets are selected. Default is false.	Boolean
defaultValue	The default value of this form field that is used as the value when the form is reset.	String
layout	Controls how the widget is positioned within the row.	Object
label	Text and styling applied to the widget's label.	Object
info	Supplemental help text or a tooltip displayed alongside the widget.	Object
tabIndex	Determines keyboard navigation order. Default is 0.	Integer
style	Sets a style for the selected widget. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Widget Types

Each widget's type property controls which additional properties become available under `props.columns.items[x].rows.items[x].widgets[x].*`. These properties define the behavior, validation rules, icons, messages, and styling for the selected widget type. Widget types include:

- None (Default)
- **Text**
- **Email**
- **URL**
- **Password**
- **Number**
- **Tel**
- **Text Area**
- **Radio**
- **Checkbox**
- **Toggle**
- **Slider**
- **Dropdown**
- **Date Picker**
- **Time Picker**
- **Date and Time Picker**
- **File Upload** **New in 8.3.8**

Text

A single-line text input for general purpose string entry, such as names, identifiers, or short notes. The text properties appear under `props.columns.items[x].rows.items[x].widgets[x].text.*`

Text Properties

Name	Description	Property Type
autoComplete	Provides browser-level autocomplete hints.	String
placeholder	Text displayed when the field is empty.	String
validation	Configures validation constraints, error messages, and visual feedback for this widget. See the **Widget Validation** section for a list of available configurations.	Object
style	Sets a style for this widget. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Email

A single-line input optimized for capturing email addresses, with option support for multiple comma-separated entries. The email widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].email.*`

Email Properties

Name	Description	Property Type
autoComplete	Provides autocomplete hints suitable for email entry.	String
multiple	Allows multiple comma-separated email entries.	Boolean
placeholder	Text displayed when the field is empty.	String
validation	Configures validation constraints, error messages, and visual feedback for this widget. See the **Widget Validation** section for a list of available configurations.	Object
style	Sets a style for this widget. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

URL

A field for entering and validating URLs, typically used for web links or resource references. The URL widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].url.*`

URL Properties

Name	Description	Property Type
autoComplete	Provides autocomplete hints suitable for URL entry.	String
placeholder	Text displayed when the field is empty.	String
validation	Configures validation constraints, error messages, and visual feedback for this widget. See the **Widget Validation** section for a list of available configurations.	Object
style	Sets a style for this widget. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Password

A masked text input used for sensitive values such as passwords or authentication tokens. Password widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].password.*`

Password Properties

Name	Description	Property Type
autoComplete	Provides autocomplete hints for password entry.	String
placeholder	Text displayed when the field is empty.	String
validation	Configures validation constraints, error messages, and visual feedback for this widget. See the **Widget Validation** section for a list of available configurations.	Object
style	Sets a style for this widget. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Number

A numeric input that restricts entry to numbers and supports min, max, and step validation. The number widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].number.*`

> **DECIMAL INPUT**
>
> Decimal input requires `constraints.step.enabled` to be enabled. See **Widget Validation** for details on configuration.

Number Properties

Name	Description	Property Type
autoComplete	Provides autocomplete hints suitable for numeric entry.	String
placeholder	Text displayed when the field is empty.	String
validation	Configures validation constraints, error messages, and visual feedback for this widget. See the **Widget Validation** section for a list of available configurations.	Object
style	Sets a style for this widget. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Tel

A phone-number–oriented input that supports pattern validation for telephone formats. The tel widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].tel.*`

Tel Properties

Name	Description	Property Type
autoComplete	Provides autocomplete hints for phone number entry.	String
placeholder	Text displayed when the field is empty.	String
validation	Configures validation constraints, error messages, and visual feedback for this widget. See the **Widget Validation** section for a list of available configurations.	Object
style	Sets a style for this widget. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Text Area

A multi-line input for longer free-form text such as comments or descriptions. The text area widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].text-area.*`

Text Area Properties

Name	Description	Property Type
autoComplete	Provides autocomplete hints for multi-line text.	String
placeholder	Text displayed when the text area is empty.	String
spellCheck	Enables or disables browser spell checking.	Boolean
wrap	Determines how text wraps within the text area.	String
validation	Configures validation constraints, error messages, and visual feedback for this widget. See the **Widget Validation** section for a list of available configurations.	Object
style	Sets a style for this widget. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Radio

A group of mutually exclusive options where the user selects exactly one value. The radio widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].radio.*`

Radio Properties

Name	Description	Property Type
options	Defines the list of available radio options, including their labels and returned values. • **text**: The label text displayed for the option as a string. • **value**: The unique value of the radio option to be evaluated when selected. May be any type.	Array
layout	Layout configuration for the group of radio options.	Object
align	Alignment of radio options within the group.	String
justify	Spacing behavior between options.	String
textPosition	Position of the label text relative to the radio icon.	String
selectedIcon	Defines the icon shown when a radio option is selected, including its path, colors, and styles. • **path**: The icon path used for the selected state as a string. • **color.enabled**: The color used when the icon is enabled as a string. • **color.disabled**: The color used when the icon is disabled as a string. • **style**: A style object applied to the selected icon. A full menu of **style options** is available and you can also specify a **style class**.	Object
unselectedIcon	Defines the icon shown when a radio option is not selected, including its path, enabled and disabled colors, and styles. • **path**: The icon path used for the unselected state as a string. • **color.enabled**: The color used when the option is enabled as a string. • **color.disabled**: The color used when the option is disabled as a string.	Object

Name	Description	Property Type
	• **style**: A style object applied to the unselected icon. A full menu of **style options** is available and you can also specify a **style class**.	
validation	Configures validation constraints, error messages, icons, and style for this widget. See the **Widget Validation** section for a full list of available validation properties.	Object
style	A style object applied to the widget. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object
optionStyle	A style object applied to each radio option. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Checkbox

A group of independent options where each item can be checked or unchecked. The checkbox widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].checkbox.*`.

Checkbox Properties

Name	Description	Property Type
type	Controls the checkbox behavior and visual style.	String
options	Defines the set of checkbox options, including labels, values, and whether each supports a tri-state mode. • **text**: The label text to display for the checkbox option as a string. • **value**: The unique value associated with the checkbox option, evaluated when the option is selected and may be any type. • **triState**: Allows the option to represent checked, unchecked, or indeterminate as a boolean.	Array
layout	Layout configuration for the group of checkbox options.	Object
align	Alignment of the checkbox options within the group.	String
justify	Spacing behavior between checkbox options.	String
textPosition	Position of the label text relative to the checkbox icon.	String
checkedIcon	Defines the icon shown when the checkbox is in the checked state, including its path, enabled and disabled colors, and styles. • **path**: The icon path used when the checkbox is checked as a string. • **color.enabled**: The color used when the icon is enabled as a string. • **color.disabled**: The color used when the icon is disabled as a string. • **style**: A style object applied to the checked icon. A full menu of **style options** is available and you can also specify a **style class**.	Object
indeterminateIcon	Defines the icon used when the checkbox is in an indeterminate state, along with its colors and style.	Object

Name	Description	Property Type
	• **path**: The icon path used when the checkbox is in an indeterminate state as a string. • **color.enabled**: The color used when the icon is enabled as a string. • **color.disabled**: The color used when the icon is disabled as a string. • **style**: A style object applied to the indeterminate icon. A full menu of **style options** is available and you can also specify a **style class**.	
uncheckedIcon	Defines the icon shown when the checkbox is not selected, including path, enabled and disabled colors, and styles. • **path**: The icon path used when the checkbox is unchecked as a string. • **color.enabled**: The color used when the icon is enabled as a string. • **color.disabled**: The color used when the icon is disabled as a string. • **style**: A style object applied to the unchecked icon. A full menu of **style options** is available and you can also specify a **style class**.	Object
validation	Configures validation constraints, error messages, icons, and style for this widget. See the **Widget Validation** section for a full list of available validation properties.	Object
style	A style object applied to the checkbox group. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object
optionStyle	A style object applied to each checkbox option. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Toggle

A switch-style selection control used for boolean or labeled on/off choices. The toggle widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].toggle.*`

Toggle Properties

Name	Description	Property Type
type	Controls the toggle behavior and visual style. Options include boolean and string types.	String
options	Defines the set of toggle options, including the displayed text and the value returned when selected. • **text**: The label text to display for the toggle option as a string. • **value**: The unique value associated with the toggle option, evaluated when the option is selected and may be any type.	Array
layout	Layout configuration for the group of toggle options.	Object
align	Alignment of the toggle options within the group.	String
justify	Spacing behavior between toggle options.	String
textPosition	Position of the label text relative to the toggle control.	String
color	Defines the colors used for selected and unselected toggle options. • **selected**: The color used when the toggle option is selected as a string. • **unselected**: The color used when the toggle option is not selected as a string.	Object
validation	Configures validation constraints, error messages, icons, and style for this widget. See the **Widget Validation** section for a full list of available validation properties.	Object
style	A style object applied to the toggle group. Full menu of **style options** is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	Object

Name	Description	Property Type
optionStyle	A style object applied to each toggle option. Full menu of **style options** is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	Object

Slider

A draggable control used to select a numeric value within a defined range. The slider widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].slider.*`.

Slider Properties

Name	Description	Property Type
labels	Controls how tick labels appear along the slider track, including whether they show and how often they appear. • **show**: Determines whether tick labels are displayed along the slider track as a boolean. • **interval**: The interval between displayed label values as a number.	Object
max	The maximum value allowed by the slider.	Number
min	The minimum value allowed by the slider.	Number
step	The increment applied when adjusting the slider value.	Number
handleColor	The color applied to the slider handle.	String
railColor	The color applied to the background rail track.	String
trackColor	The color applied to the filled portion of the slider track.	String
style	A style object applied to the slider. Full menu of **style options** is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	Object

Dropdown

A collapsible list of options used to select one or multiple values from a predefined list. The dropdown widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].dropdown.*`.

Dropdown Properties

Name	Description	Property Type
options	Defines the list of dropdown options, including their labels, values, and disabled state. • **value**: The underlying value associated with the option and may be any type. • **label**: The text displayed for the option as a string. • **isDisabled**: Determines whether the option is selectable as a boolean.	Array
multiselect	Allows multiple options to be selected.	Boolean
wrapMultiSelectValues	Wraps selected values instead of truncating them.	Boolean
placeholder	Controls the placeholder text, color, and icon shown when no selection has been made. • **text**: Placeholder text shown when no value is selected as a string. • **color**: The color applied to the placeholder text as a string. • **icon**: The icon configuration for the placeholder as an object.	Object
search	**New in 8.3.8** Allows users to enter text to search for results. Configurable search options include: • **enabled**: Whether options are searchable by typing text into the field. Default is true. Value is boolean. • **matching**: Whether the search string must match from the start or may match any position of an option. Value is string. • **noResultsText**: The text to display in the dropdown when no options match the search. Default is "No results found". Value is string. • **searchParam**: The text being searched for. This property is read-only, and any input in this field will have no effect and will be overwritten. Value is string.	Object

Name	Description	Property Type
showClearIcon	**New in 8.3.8** Whether to render a button allowing a user to clear the selection. Default is false.	Boolean
allowCustomOptions	**New in 8.3.8** Whether a user may enter a custom value to be submitted. Default is false. While set to True, typing a value that doesn't match one of the existing options from the session will provide the user with a Create option. Selecting Create will set `props.value` to the new option. Creating a custom option in this way does not change the value of `props.options`. The custom option is simply a means to allow the user to type a custom value into the component.	Boolean
textAlign	Alignment of the displayed text within the field.	String
minMenuHeight	The minimum height for the dropdown menu.	Number
maxMenuHeight	The maximum height for the dropdown menu.	Number
validation	Configures validation constraints, error messages, icons, and style for this widget. See the **Widget Validation** section for a full list of available validation properties.	Object
style	A style object applied to the dropdown. Full menu of **style options** is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	Object
optionStyle	A style object applied to each dropdown option. Full menu of **style options** is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	Object

Date Picker

A control for selecting calendar dates with formatting and locale options. The date picker widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].date-picker.*`.

Date Picker Properties

Name	Description	Property Type
defaultToNow	Uses the current date as the initial value when enabled.	Boolean
displayType	Controls how the date picker appears, such as inline or dialog.	String
minDate	The earliest selectable date.	String
maxDate	The latest selectable date.	String
format	The format string used to display the date.	String
placeholder	The text displayed when no date is selected.	String
locale	The locale used for date formatting and localization.	String
dismissOnSelect	Closes the date picker when a date is selected.	Boolean
validation	Configures validation constraints, error messages, icons, and style for this component. See the **Widget Validation** section for a full list of available validation properties.	Object
style	A style object applied to the date picker. Full menu of **style options** is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	Object
modalStyle	A style object applied to the date picker modal or popup. Full menu of style options is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a style class.	Object

Time Picker

A control for selecting times of day, with optional spinner or typed entry. The time picker widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].time-picker.*`.

Time Picker Properties

Name	Description	Property Type
defaultToNow	Uses the current time as the initial value when enabled.	Boolean
minTime	The earliest selectable time.	String
maxTime	The latest selectable time.	String
showSpinners	Shows increment and decrement controls for adjusting time.	Boolean
format	The format string used to display the time.	String
locale	The locale used for time formatting and localization.	String
style	A style object applied to the time picker. Full menu of **style options** is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	Object

Date and Time Picker

A combined control for selecting both a date and a time. The date and time Picker widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].date-time-picker.*`.

Date and Time Picker Properties

Name	Description	Property Type
defaultToNow	Uses the current date and time as the initial value when enabled.	Boolean
displayType	Controls how the picker is displayed, such as inline or dialog.	String
minDate	The earliest selectable timestamp.	String
maxDate	The latest selectable timestamp.	String
format	The format string used to display the date and time.	String
placeholder	The text displayed when no value is selected.	String
locale	The locale used for formatting and localization.	String
dismissOnSelect	Closes the picker when a value is selected.	Boolean
validation	Configures validation constraints, error messages, icons, and style for this widget. See the **Widget Validation** section for a full list of available validation properties.	Object
style	A style object applied to the date and time picker. Full menu of **style options** is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a **style class**.	Object
modalStyle	A style object applied to the picker modal or popup. Full menu of style options is available for text, background, margin and padding, border, shape, and miscellaneous. You can also specify a style class.	Object

File Upload

New in 8.3.8

A control for selecting and uploading one or more files as part of the form. The file upload widget properties appear under `props.columns.items[x].rows.items[x].widgets[x].file-upload.*`.

File Upload Properties

Name	Description	Property Type
validation	Configures validation constraints, error messages, and visual feedback for this widget. See the **Widget Validation** section for a list of available configurations.	Object
style	Sets a style for this widget. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	Object

Widget Validation

The `constraints`, `icon`, `messages`, and `style` groups make up the validation settings for widgets. Although these groups appear across multiple widget types, not every sub-property is shared. Refer to the Supported Widgets column to see which widgets implement each sub-property.

Name	Description	Property Type	Supported Widgets
constraints.maxLength.enabled	Enables validation that checks whether the input exceeds the maximum allowed length.	Boolean	• text • email • password • url • tel • text-area
constraints.maxLength.value	Specifies the maximum number of characters allowed.	Number	• text • email • password • url • tel • text-area
constraints.minLength.enabled	Enables validation that checks whether the input meets the minimum required length.	Boolean	• text • email • password • url • tel • text-area
constraints.minLength.value	Specifies the minimum number of characters required.	Number	• text • email • password • url • tel • text-area

Name	Description	Property Type	Supported Widgets
constraints.pattern.enabled	Enables validation that checks whether the input must match a regular expression.	Boolean	• text • email • password • url • tel
constraints.pattern.value	Specifies the regular expression the input must match.	String	• text • email • password • url • tel
constraints.required.enabled	Enables validation that checks whether a value has been provided.	Boolean	All types
constraints.min	Specifies the lowest numeric value allowed.	Number	number
constraints.max	Specifies the highest numeric value allowed.	Number	number
constraints.step.enabled	**New in 8.3.4** Enables validation that enforces the specific interval between numeric values as set by the `step.value` property. **Note:** When constraints are enabled, this property must also be enabled	Boolean	number

Name	Description	Property Type	Supported Widgets
	to allow decimal values. Otherwise, input is limited to integers.		
constraints.step.value	**New in 8.3.4** Defines the interval between allowed numeric values. Set to `any` to allow any value.	String	number
constraints.fileSizeLimit.value	**New in 8.3.8** Specifies the maximum size, in megabytes, allowed for each uploaded file.	Number	file-upload
constraints.maxUploads.enabled	**New in 8.3.8** Enables validation that limits the number of files that can be uploaded.	Boolean	file-upload
constraints.maxUploads.value	**New in 8.3.8** Specifies the maximum number of files that can be uploaded.	Number	file-upload
constraints.supportedFileTypes.enabled	**New in 8.3.8** Enables validation that restricts uploads to specific file types.	Boolean	file-upload
constraints.supportedFileTypes.value	**New in 8.3.8** Specifies the list of allowed file types.	Array	file-upload

Name	Description	Property Type	Supported Widgets
icon.path	Specifies the icon displayed when validation feedback becomes active.	String	All types
icon.color	Specifies the color applied to validation icons.	String	All types
icon.style	Specifies the style applied to validation icons.	Object	All types
messages.valueMissing	Displays when a required field has no value.	String	All types
messages.patternMismatch	Displays when input does not match the required pattern.	String	• text • email • password • url • tel
messages.tooLong	Displays when input exceeds the maximum allowed length.	String	• text • email • password • url • tel • text-area
messages.tooShort	Displays when input does not meet the minimum required length.	String	• text • email • password • url

Name	Description	Property Type	Supported Widgets
			• tel • text-area
messages.typeMismatch	Displays when the value does not match the expected type.	String	• email • url
messages.rangeUnderflow	Displays when a numeric value is lower than the minimum allowed.	String	• number • slider
messages.rangeOverflow	Displays when a numeric value is higher than the maximum allowed.	String	• number • slider
messages.fileSizeLimit	**New in 8.3.8** Displays when an uploaded file exceeds the maximum allowed file size.	String	file-upload
messages.maxUploads	**New in 8.3.8** Displays when the amount of uploaded files exceeds the maximum number of allowed uploads.	String	file-upload
messages.supportedFileTypes	**New in 8.3.8** Displays when an uploaded file is not an allowed file type.	String	file-upload
validation.style	Specifies styles applied to validation	Object	All types

Name	Description	Property Type	Supported Widgets
	messages and indicators.		

> **DECIMAL INPUT IN NUMBER WIDGETS**
>
> For the Number widget type, enabling constraints without enabling `constraints.step.enabled` limits input to integer values. To allow decimal values, enable `constraints.step.enabled` and set `constraints.step.value` to the interval you want to allow, or to `any`.

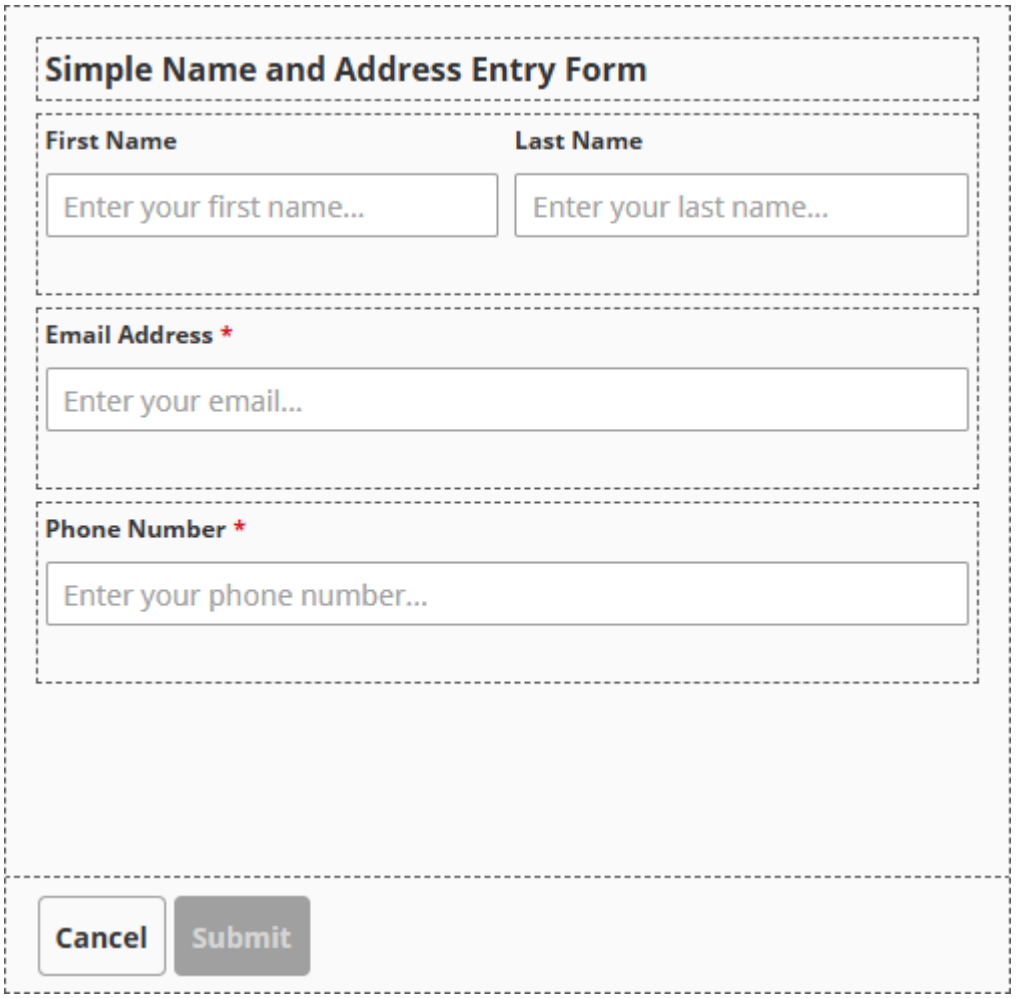
Scripting

See the **Perspective - Form Scripting** page for the full list of scripting functions available for this component.

Examples

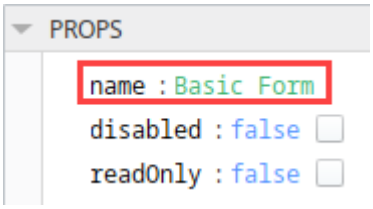
Create a Basic Form Layout with Validation

When a Form component is first added to a view, it will have a single column containing two rows for the header and input. This example demonstrates how to further configure multiple rows and validation for email and phone number fields.



To configure the Form component with validation:

1. Drag a **Form** component onto your view.
2. Select the Form component and set **props.name** to `Basic Form` in the Property Editor.



3. Edit the header row.
    i. Navigate to **columns.items[0].rows.items[0].widgets**.
    ii. Expand **0**, then set the following values:
        - **id** to `header`
        - **label.text** to `Simple Name and Address Entry Form`
        - (Optional) Delete or update **info.text**. In this example, each **info.text** value has been deleted.



4. Edit the existing input row for first and last name.

    i. Navigate to **columns.items[0].rows.items[1].widgets**.

    ii. Expand **0**, then set the following values:

    - **id** to `firstName`
    - **type** to `text`
    - **text.placeholder** to `Enter your first name...`
    - **label.text** to `First Name`



iii. Add a second widget to the same row.

a. Click the **Add** ➕ icon next to **widgets**.

b. Expand **1**, then set the following values:

- **id** to `lastName`
- **type** to `text`
- **text.placeholder** to `Enter your last name...`
- **label.text** to `Last Name`



5. Add a row for the email field.

   i. Navigate to **columns.items[0].rows.items** and click the **Add** ➕ icon next to **items**.

   ii. Navigate to **rows.items[2].widgets**.

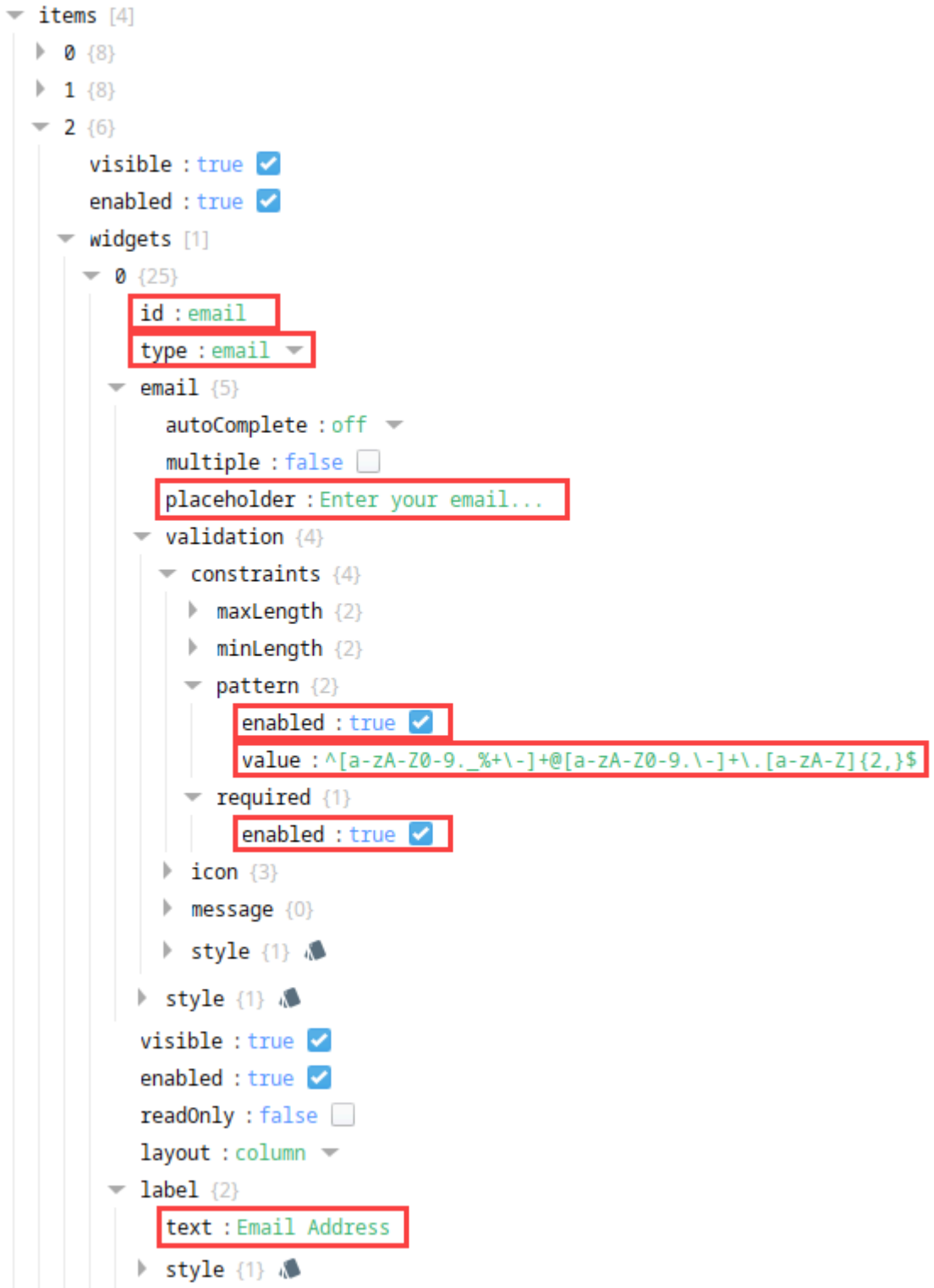
   iii. Expand **0**, then set the following values:

   - **id** to `email`
   - **type** to `email`
   - **text.placeholder** to `Enter your email...`
   - **label.text** to `Email Address`

   iv. Expand **email.validation.constraints.pattern**, then set the following values:

   - **enabled** to `true`
   - **value** to `^[a-zA-Z0-9._%+\-]+@[a-zA-Z0-9.\-]+\.[a-zA-Z]{2,}$`

   v. Expand **email.validation.constraints.required**, then set **enabled** to `true`.



6. Add a row for the phone number field.

   i. Navigate to **columns.items[0].rows.items** and click the **Add** ➕ icon next to **items**.

   ii. Navigate to **rows.items[3].widgets**.

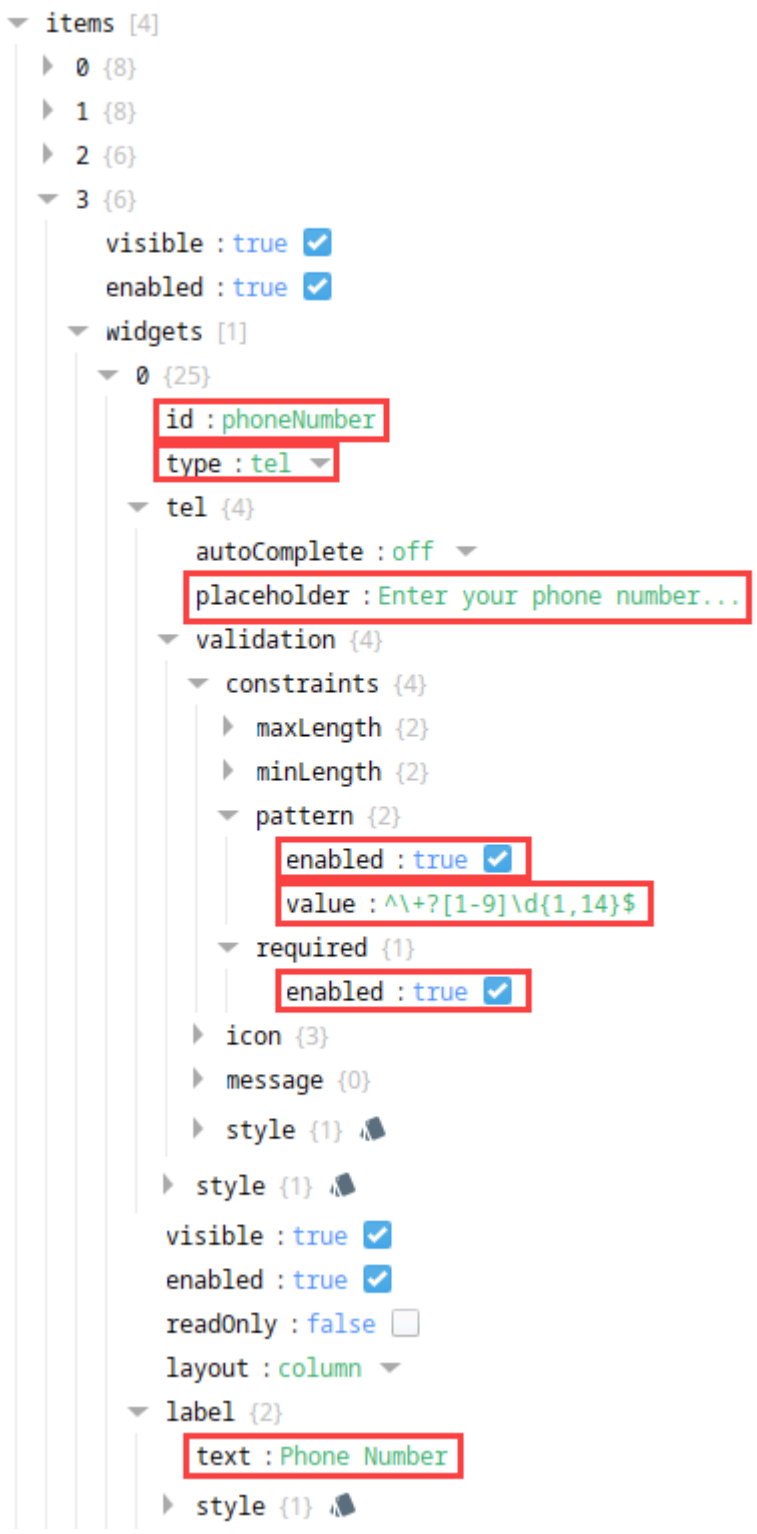
   iii. Expand 0, then set the following values:

   - **id** to `phoneNumber`
   - **type** to `tel`
   - **text.placeholder** to `Enter your phone number...`
   - **label.text** to `Phone Number`

   iv. Expand **tel.validation.constraints.required**, then set `enabled` to `true`.

v. Expand **tel.validation.constraints.pattern**, then set the following values:

- **enabled** to `true`
- **value** to `^\+?[1-9]\d{1,14}$`



Properties Used

Property	Val
props.name	Bas
props.columns.items[0].rows.items[0].widgets[0].id	hea
props.columns.items[0].rows.items[0].widgets[0].label.text	Sim and Ent
props.columns.items[0].rows.items[1].widgets[0].id	first
props.columns.items[0].rows.items[1].widgets[0].type	text
props.columns.items[0].rows.items[1].widgets[0].text.placeholder	Ent nan
props.columns.items[0].rows.items[1].widgets[0].label.text	Firs
props.columns.items[0].rows.items[1].widgets[1].id	last
props.columns.items[0].rows.items[1].widgets[1].type	text
props.columns.items[0].rows.items[1].widgets[1].text.placeholder	Ent nan
props.columns.items[0].rows.items[1].widgets[1].label.text	Las
props.columns.items[0].rows.items[2].widgets[0].id	ema
props.columns.items[0].rows.items[2].widgets[0].type	ema
props.columns.items[0].rows.items[2].widgets[0].text.placeholder	Ent ema
props.columns.items[0].rows.items[2].widgets[0].label.text	Em
props.columns.items[0].rows.items[2].widgets[0].email.validation.constraints.pattern.enabled	true

Property	Val
props.columns.items[0].rows.items[2].widgets[0].email.validation.constraints.pattern.value	`^[a` `9._` `zA-` `9.\` `zA-`
props.columns.items[0].rows.items[2].widgets[0].email.validation.constraints.required.enabled	true
props.columns.items[0].rows.items[3].widgets[0].id	pho
props.columns.items[0].rows.items[3].widgets[0].type	tel
props.columns.items[0].rows.items[3].widgets[0].text.placeholder	Ent pho num
props.columns.items[0].rows.items[3].widgets[0].label.text	Pho
props.columns.items[0].rows.items[3].widgets[0].tel.validation.constraints.required.enabled	true
props.columns.items[0].rows.items[3].widgets[0].tel.validation.constraints.pattern.enabled	true
props.columns.items[0].rows.items[3].widgets[0].tel.validation.constraints.pattern.value	`^\+` `9]\`

Try it Yourself: Form Component Contingent Actions Guided Tutorial

Open Training Exercise

Perspective - Form Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Form** component. Scripting allows you to customize form behavior, handle validation dynamically, and respond to user actions.

Component Events

The **Perspective Event Types Reference** page describes all possible event types for Perspective components. Not all event types apply to every component. For more information on configuring events, see **Component Events and Actions**.

onChange

This event triggers whenever a form's input value changes.

Properties

Property	Type	Description
event.inputKey	String	The unique key identifying the input that triggered the change.
event.newValue	Object	The new value of the input after the change.
event.oldValue	Object	The previous value of the input before the change occurred.
event.index	Integer	The index of the input if multiple inputs share the same key, such as in a repeater or table. Optional.

onSuccess

This event triggers when a form submission completes successfully.

Properties

Property	Type	Description
event	Object	The success response object returned after submission.

See **Form Submission** for handling form data on submission.

onError

This event triggers when a form submission fails.

Properties

Property	Type	Description
event	Object	The error response object returned from the submission.

See **Form Submission** for handling errors.

onCancelActionPerformed

If configured, this component event fires when the Form's cancel action is clicked.

Properties

Property	Type	Description
event	Object	A JSON object containing all form input data to be cancelled.

onSubmitActionPerformed

If configured, this component event fires when the Form's submit action is clicked.

Properties

Property	Type	Description
event	Object	A JSON object containing all form input data to be submitted.

Component Functions

.setCustomValidity(message, inputKey, index)

Description

Sets a custom validation message for a specific input field in the form.

Parameters

Parameter	Type	Description
message	String	The custom validation message to display for this input.
inputKey	String	The unique key identifying the input field.
index	Integer	The index of the input if multiple inputs share the same key. Optional.

Return

None

Example

To set a custom validation message for an email input field:

```
self.getSibling("Form").setCustomValidity("Please provide a valid email address.", "email")
```

> **INFO**
>
> This message will persist until the field is corrected or the form is reset.

.reset()

Description

Resets all form inputs to their default values.

Parameters

None

Return

None

Example

To reset the form inputs to their default state:

```
self.getSibling("Form").reset()
```

.submit(preventInvalid)

Description

Submits the form, triggering **onSuccess** if successful or **onError** if submission fails.

Parameters

Parameter	Type	Description
preventInvalid	Boolean	When true, prevents the form from submitting if it is invalid. Optional.

Return

None

Example

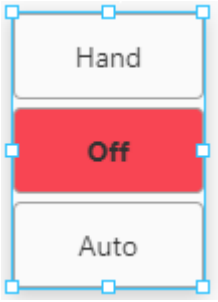
To submit the form with validation check enabled:

```
self.getSibling("Form").submit(preventInvalid=True)
```

Extension Functions

This component does not have any extension functions associated with it.

Perspective - Multi-State Button



Component Palette Icon:



Description

The Multi-State Button is really a series of two or more buttons, arranged in a column or row. Each button represents an integer-valued state. Each state defines two styles for a button: the selected style, and the unselected style. Each button is automatically displayed with the correct style based on the current state (the value of Indicator Value). When a button is pressed, its state's value is written to the Control Value.

When the Multi-State Button is dragged to a container, it is pre-configured with 'defaultSelectedStyle' and 'defaultUnselectedStyle properties'. These styles can be changed or deleted.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
controlValue	Bind this to the Tag that controls the state. (Typically, this is bound to the same location as the indicatorValue property.)	value: numeric
indicatorValue	Bind this to the Tag that indicates the current state. (Typically, this is bound to the same location as the controlValue property).	value: numeric
states	The value that will be written to **controlValue** when any of the buttons are clicked. Shows a list of the possible states for the component. You can add, remove, and the change the order of each state listed. Each state has two default visual styles applied for each button: Selected Style and Unselected Style. The Multi-State Button has default visual styles defined for both the selectedStyle and unselectedStyle. (Refer to 'defaultSelectedStyle' and 'defaultUnselectedStyle' properties in this table). • **text**: Text displayed on the button. Value is string. • **value**: Value assigned to the state. Value is numeric. • **selectedStyle**: Style settings for the button when it is selected. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **unselectedStyle**: Style settings for the button when it is not selected. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **tooltipText**: Determines what test should appear when mouse cursor hovers over the button associated with this property. If blank, no tooltip will appear. Value is string. **Note**: This property is not present by default, and must be added manually.	object

Name	Description	Property Type
orientation	Physical position of the button: Column or Row.	boolean
defaultSelectedStyle	Default selected style. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
defaultUnselectedStyle	Default styles for unselectedStyles when any of the buttons are *not* selected. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
primary	Toggles between the default primary and secondary button style.	value: boolean
enabled	If true, the user is able to interact with the buttons.	value: boolean
buttonGap	Space, in pixels, between each button in a group.	value: numeric
endButtonCornerRadius	Amount to round the end of the corners of the first and last button.	value: numeric
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective

component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

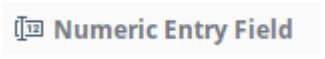


Property	Value
props.indicatorValue	(property binding)
props.states.0.text	Open
props.states.0.value	2
props.states.0.selectedStyle.backgroundColor	#FFF809
props.states.1.text	Close
props.states.1.value	0
props.states.1.selectedStyle.backgroundColor	#FF8C00
props.states.2.text	Auto
props.states.2.value	1
props.states.2.selectedStyle.backgroundColor	#62ED2A
props.states.3.text	Bypass
props.states.3.value	4
props.states.3.selectedStyle.backgroundColor	#FF0000
props.orientation	row
props.buttonGap	8
props.endButtonCornerRadius	2

Perspective - Numeric Entry Field



Component Palette Icon:



Description

The Numeric Entry Field is similar to the standard Text Field, except that it is specialized for use with numbers. When the 'enabled' property is set to true, it allows users to alter the value on the component. There are three different modes for how users can edit the value in the component: direct, protected or by clicking an edit button. To change the value, click once in the field for 'direct' mode, double click for 'protected' mode, and click on the Edit icon for the 'button' mode. When done, press enter or leave the field, and the field becomes editable again. When the 'enabled' property is false, the field is not editable even when it receives input focus.

The Numeric Entry Field component has three pre-configured **variants**:

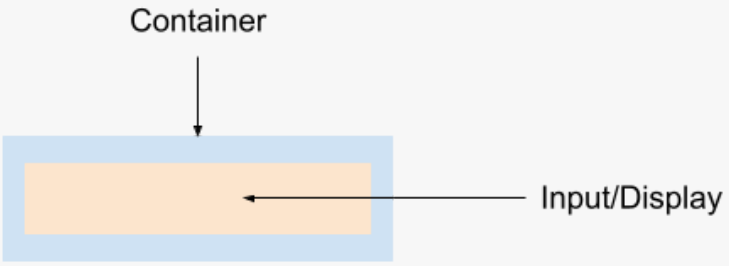
- Direct - Default design of the field.
- Protect - Requires a double-click or long-press to enter edit mode.
- Button - Clicking the button brings up a popup window, allowing the user to edit the value from the popup, or cancel the edit.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	Value as number or numeric string to display.	value: numeric
format	The formatting string to be applied to the input value. Options are currency, number, integer, four decimal precision, percent, scientific, accounting, financial, currency, currency (rounded), duration, abbreviation, or ordinal. A list of format specifiers can be found at **http://numeraljs.com/#format**. The Numeric Entry Field supports locale-specific formatting, allowing changes to the session's locale to update how numbers are formatted on the Numeric Entry Field. Note that the localization conversion occurs automatically *after* the initial format specifiers are applied.	value: string
mode	Determines how users will edit the value in the component. The following modes are available: • Direct: Requires no special action to enter edit mode simply click in the field. • Protected: Requires a double-click or long-press to enter edit mode. • Button: Places an **Edit**  icon next to the Numeric Entry Field. Clicking the button brings up a popup window, allowing the user to edit the value from the popup, or cancel the edit. The value in the component may only be edited via the popup.	value: string
align	Aligns the input value right or left.	value: boolean
inputBounds	Max and min bounds configuration. • **maximum** The max allowable value. Value is numeric. • **minimum**: The min allowable value. Value is numeric. • **invalidStyle**: Sets an invalid style when the min or max values are out of bounds for this component. Modify the invalidStyle using the style properties. Full menu of **style options** is available for text, background, margin and	object

Name	Description	Property Type
	padding, border, shape and miscellaneous. You can also specify a **style class**.	
placeholder	Text to be displayed when value is empty.	value: string
spinner	Optional spinner configuration. • **enabled**: If enabled, a spinner will appear when the field is selected by the user. Value is boolean. • **increment**: The increment the spinner uses to increase or decrease the value. Value is numeric.	object
tooltipText	Mousing over this button will display a tooltip with this text, if present.	value: string
enabled	Indicates if user should be allowed to alter the value. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a **script action** to a System Event, such as an onStartup event, the script will fire when the page is loaded. Events that require user interaction, such as onClick events, will not fire with the exception of Pointer Events.	value: boolean
containerStyle	Sets a style for the outer area of the component. The image below represents a low fidelity representation of the component. The containerStyle property determines the look of the outer "container" area of the component, making it ideal for adding borders, margins around the entire component, and padding between the container and input/display area.	object

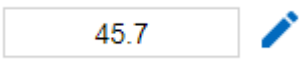
Name	Description	Property Type
	 Changes that should be made to the input/display area, such as changing the font on the displayed value, should be made on the style property.	
style	Sets a style for the "inner" numeric display/input in this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Examples

Example 1



This example shows the component set to "button" mode, providing a button to click on when entering a new value.

Property	Value
props.value	45.678
props.format	#0.0
props.mode	button
props.align	center
props.placeholder	setpoint

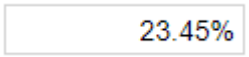
Example 2



This examples demonstrates the placeholder property, showing a default entry in cases where the value is null.

Property	Value
props.value	null
props.format	#0.0
props.mode	button
props.align	center
props.placeholder	setpoint

Example 3



This example demonstrates the format property, allowing custom formatting to be applied to the value in the component.

Property	Value
props.value	0.2345
props.format	0.0#%
props.mode	direct
props.align	right
props.placeholder	setpoint

Perspective - One-Shot Button



Component Palette Icon:



Description

The One-Shot Button is designed to send off a write request, and wait for a response, disabling the button until something resets the 'value' property on the component.

When the 'value' property and the 'setValue' property are equal, the component transitions to the writing state. Once 'value' and 'setValue' are no longer equal, the button returns to the ready state.

The One-Shot Button has three pre-configured **variants**:

- Primary - Default design of the button.
- Secondary - A secondary design for the button.
- Require Confirm - Default design of the button but requires confirmation from user before action is submitted.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	The current value displayed on the component. Should be bound bi-directionally to a tag. Default is 0.	value: numeric, boolean, string, or null.
setValue	The value to set when the button is pushed. Default is 1.	value: numeric, boolean, string, or null.
primary	Toggle between the default primary and secondary button style. Default is true.	value: boolean
enabled	Whether the user can interact with the One-Shot Button. If disabled, the component cannot be used. Default is true.	value: boolean
readyState	Displays the readyState value on the component. • **text**: The text of the button while its value is not being written. Value is string. • **style**: Modify readyState style using the style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **icon**: The Icon is an image path used to augment the readyState of the component by placing an image next to it. ◦ **path**: Shorthand path to icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. ◦ **color**: Color of the icon. Can also assign color in "fill" of the style property. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	object

Name	Description	Property Type
	◦ **style**: Modify the readyState style using the style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
writingState	Displays the writingState value on the component. • **text**: The text of the button while its value is being written. Value is string. • **style**: Modify readyState style using the style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **icon**: The Icon is an image path used to augment the readyState of the component by placing an image next to it. ◦ **path**: Shorthand path to icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. ◦ **color**: Color of the icon. Can also assign color in "fill" of the style property. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. ◦ **style**: Modify the readyState style using the style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
confirm	If enabled, a message that asks the user to approve the requested operation. If the user doesn't confirm, then the value property will not update.	object

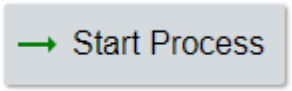
Name	Description	Property Type
	Note: When configuring a script on this component while confirm is enabled, it is generally recommend that the script is placed on the `onActionPerformed` component event, since that event will wait for user confirmation. • **enabled**: If true, a confirmation box will be shown. Default is false. Value is boolean. • **text**: Message to show user if confirmation is enabled. Default is "Are you sure?". Value is string.	
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
disabledStyle	Sets a style for this component when it is **disabled**. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

> **NOTE**
>
> If you are using both the style and disabledStyle properties, keep in mind that CSS dictates that style classes will be rendered in alphabetical order. In other words, specific elements of a style class will be overwritten if a later style class modifies the same element.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

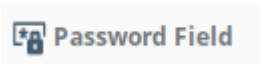


Property	Value	Style Category
props.readyState.text	Start Process	N/A
props.readyState.icon.path	material/trending_flat	N/A
props.readyState.icon.color	#008000	N/A
props.writingState.text	Starting	N/A
props.writingState.style.backgroundColor	#8AFF8A	background

Perspective - Password Field



Component Palette Icon:



Description

The Password Field component is similar to a Text Field component. It allows users to enter their password text. When the Password field is empty, you can create a placeholder that informs user to "Login". You can also enable the "allowReveal" property to allow users to view their password entry.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
text	Password text.	value: string
placeholder	Text displayed when password text is empty.	value: string
enabled	Whether the user can alter the password text. Default is true. **Note**: If the component is disabled, scripts can still run on the component. For example, if you add a **script action** to a System Event, such as an onStartup event, the script will fire when the page is loaded. Events that require user interaction, such as onClick events, will not fire with the exception of Pointer Events.	value: boolean
allowReveal	Whether the user can temporarily remove the password mask, revealing the password.	value: boolean
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example



Property	Value
props.placeholder	Enter Password
props.borderStyle	solid
props.borderColor	#D90000
props.borderWidth	3px
props.fontSize	14px
propsfontWeight	bold

Perspective - Radio Group



Component Palette Icon:



Description

The Radio Group allows you to create multiple radio buttons in a single container. The number of radio buttons in the group is determined by the number of elements in the "radios" object. Only one radio button in a group may be selected at a time. Radio groups are a good way to let the user choose just one of a number of options. If multiple selections are expected, the Checkbox or Dropdown components can be used.

The Radio Group component has three pre-configured **variants**:

- Text Right - Default layout with text on the right of the radio button.
- Text Left - Layout with text on the left of the radio button.
- Multiple - Layout with multiple radio buttons and text on the right.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	The value of the selected radio.	value: string, integer, boolean, or null
index	The index of the selected node.	value: integer
radios	List of radios that make up this group. • **enabled**: **New in 8.3.5** Whether the user is allowed to select this radio. Value is boolean. • **text**: Text to pair with this radio. Value is string. • **selected**: If 'true,' this radio is selected. Value is boolean. • **value**: The value of the radio to be evaluated when selected. Value is numeric. • **style**: **New in 8.3.5** Sets a style for this radio. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	array
orientation	Placement of the Radio Button: row or column. Default is row.	value: string
align	Align radios along the cross axis. Vertical if orientation is set to row; horizontal if orientation is set to column.	value: string
justify	Justify radios along the main axis. Horizontal if orientation is set to row; vertical if orientation is set to column.	value: string
textPosition	Where to place the label text in relation to the Radio Button: top, right, bottom, or left. Default is right.	value: string
enabled	If true, user is allowed to select a radio. Default is true.	value: boolean

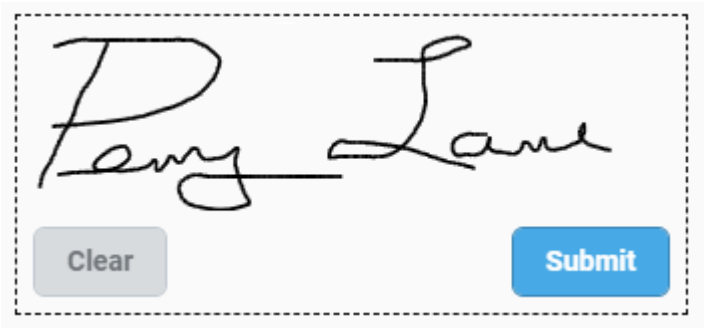
Name	Description	Property Type
	Note: If the component is disabled, scripts can still run on the component. For example, if you add a script action to the onClick event, the script will fire when the user clicks on the Radio Button.	
selectedIcon	Settings for the appearance of the radio's icon when it is selected. • **path**: Path to the icon source, in format: library/IconName. For more information on icons, see the **Images, and Icons in Perspective** page. Value is string. • **color**: Settings for the fill color for the icon. ◦ **enabled**: Color of the icon when enabled. Can be a named color. Value is string. ◦ **disabled**: Color of the icon when disabled. Can be a named color. Value is string. • **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
unselectedIcon	Settings for the appearance of the radio's icon when it is not selected. • **path**: Path to the icon source, in format: library/IconName. For more information on icons, see the **Images, and Icons in Perspective** page. Value is string. • **color**: Settings for the fill color for the icon. ◦ **enabled**: Color of the icon when enabled. Can be a named color. Value is string. ◦ **disabled**: Color of the icon when disabled. Can be a named color. Value is string. • **style**: Sets a style for the icon. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Name	Description	Property Type
radioStyle	Sets a style for the radio buttons. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Perspective - Signature Pad



Component Palette Icon:



Description

The Signature Pad component enables users to draw a signature and submit it. Submitting a signature triggers a component event, enabling Ignition to do something with the signature.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
enabled	Enables the canvas, clear button, and submit button. When enabled, component scripting functions for clearSignature and submitSignature are also enabled.	value: boolean
pad	Settings for the pad. • **pen**: Settings for the pen. ◦ **color**: Color used to draw the lines with the pen. You can set the color with a HEX, RGB, or HSL value. See also **Color Selector Reference**. ◦ **width**: Width (in pixels) of the line drawn by the pen. Value is numeric. • **canvas**: Settings for the canvas. ◦ **clearColor**: Color used to paint over the signature pad when cleared. Default is transparent. You can set the color with a HEX, RGB, or HSL value. See also **Color Selector Reference**. ◦ **style**: Sets a style for this property. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	value: object
actionBar	Settings for the actionBar. • **position**: Action bar position relative to the canvas. Options are top, bottom, left, or right. Default is bottom. Value is string. • **submitButton**: Settings for the submit button. ◦ **text**: Text to display on the button. Default is submit. Value is string. ◦ **enabled**: Enables submit button interaction. This does not affect the submitSignature component scripting function. Value is boolean. ◦ **primary**: Toggle between the default primary and secondary button style. Value is boolean. ◦ **style**: Sets a style for this property. Full menu of **style options** is available for text, background,	value: object

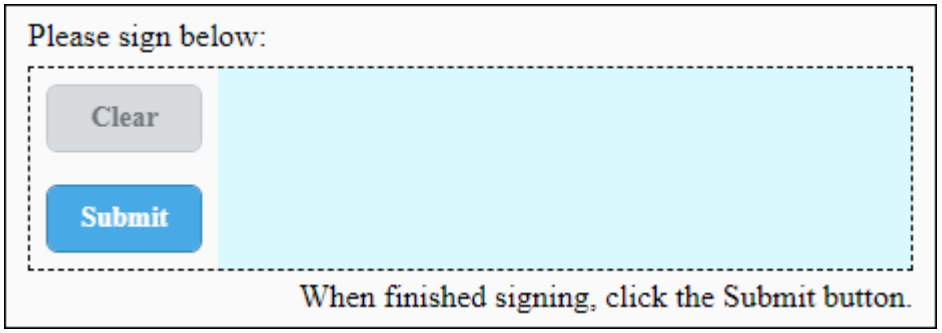
Name	Description	Property Type
	margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **clearButton**: Settings for the clear button. ◦ **text**: Text to display on the button. Default is clear. Value is string. ◦ **enabled**: Enables clear button interaction. This does not affect the submitSignature component scripting function. Value is boolean. ◦ **primary**: Toggle between the default primary and secondary button style. Value is boolean. ◦ **style**: Sets a style for this property. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. • **style**: Sets a style for this property. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	
status	Settings for the status of the component. • **touched**: True when the signature pad contains a signature. Value is boolean.	object
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Scripting

See the **Perspective - Signature Pad Scripting** page for the full list of scripting functions available for this component.

Examples

Example 1



In this example we set a few properties to customize the look of the Signature Pad. The buttons are on the left. The blue background is set with the prop.canvas.clearColor property, which enables the color to show up in our project but not get saved as part of the signature. Lastly, we put two Label components above and below the Signature Pad with signing instructions.

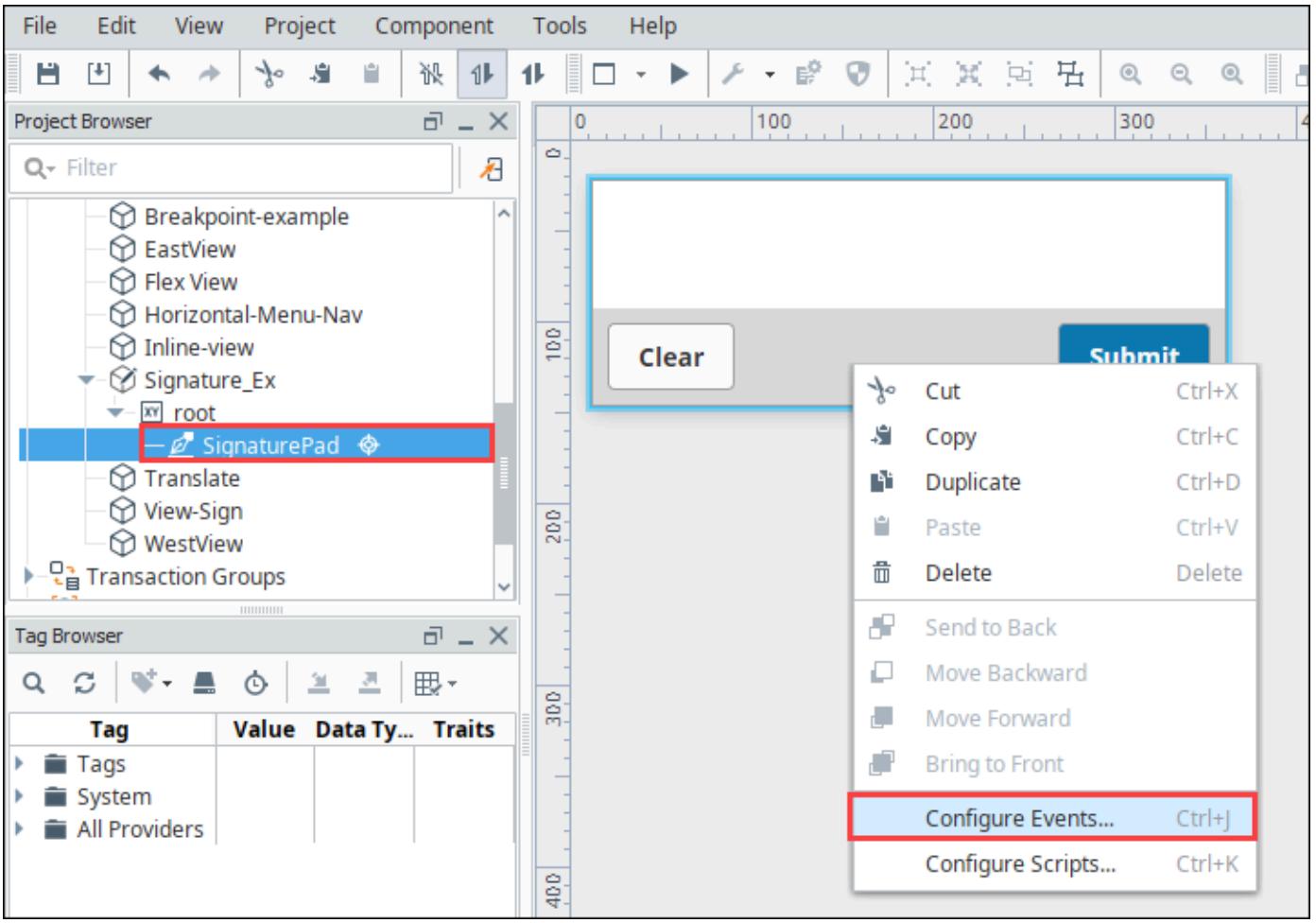
Property	Value
props.enabled	true
props.canvas.clearColor	#DAF9FF
props.actionBar.position	left
props.actionBar.submitButton.text	Submit
props.actionBar.submitButton.enabled	true
props.actionBar.submitButton.primary	true
props.actionBar.submitButton.style.fontFamily	Merriweather
props.actionBar.clearButton.text	Clear
props.actionBar.clearButton.enabled	true
props.actionBar.clearButton.primary	true
props.actionBar.clearButton.style.fontFamily	Merriweather
props.style.borderStyle	dashed
props.style.borderwidth	1pt

Example 2

The following example downloads the signature image when a user clicks the Submit button on the component.

To set this up, do the following:

1. Drag a Signature component onto a Perspective view. Make sure it's a view that has a URL.
2. Right-click on the component and select **Configure Events**.



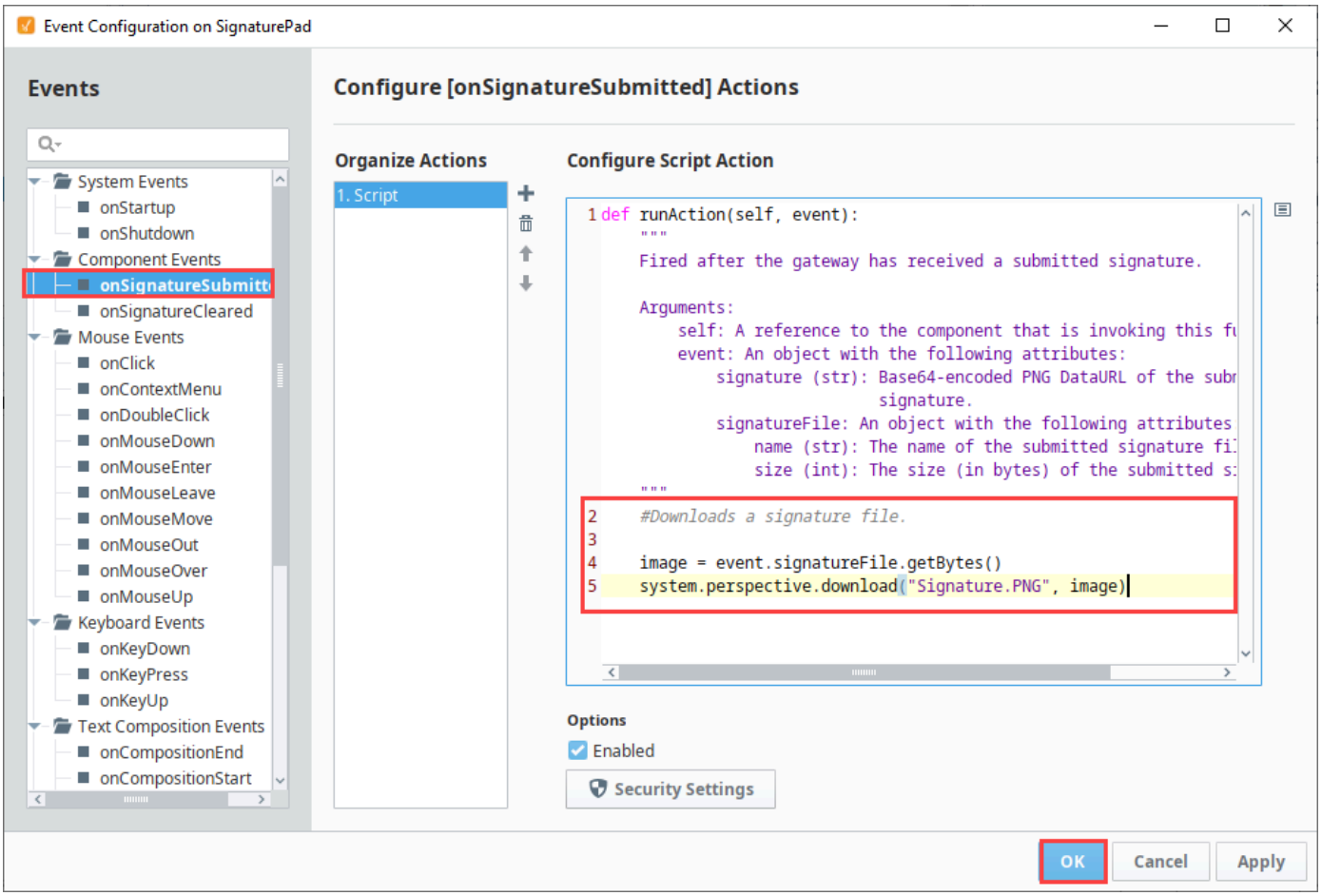
3. Select the **OnSignatureSubmitted** event.

4. Click the **Add** icon and select **Script**.

5. In the Configure Script Action box, add the following script:

```
#Downloads a signature file.

image = event.signatureFile.getBytes()
system.perspective.download("Signature.PNG", image)
```

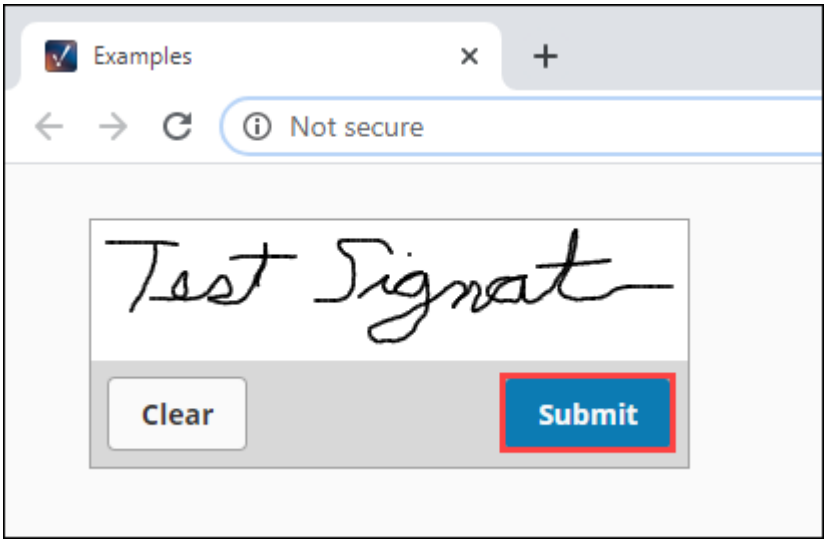


6. Click **OK** to save the script.

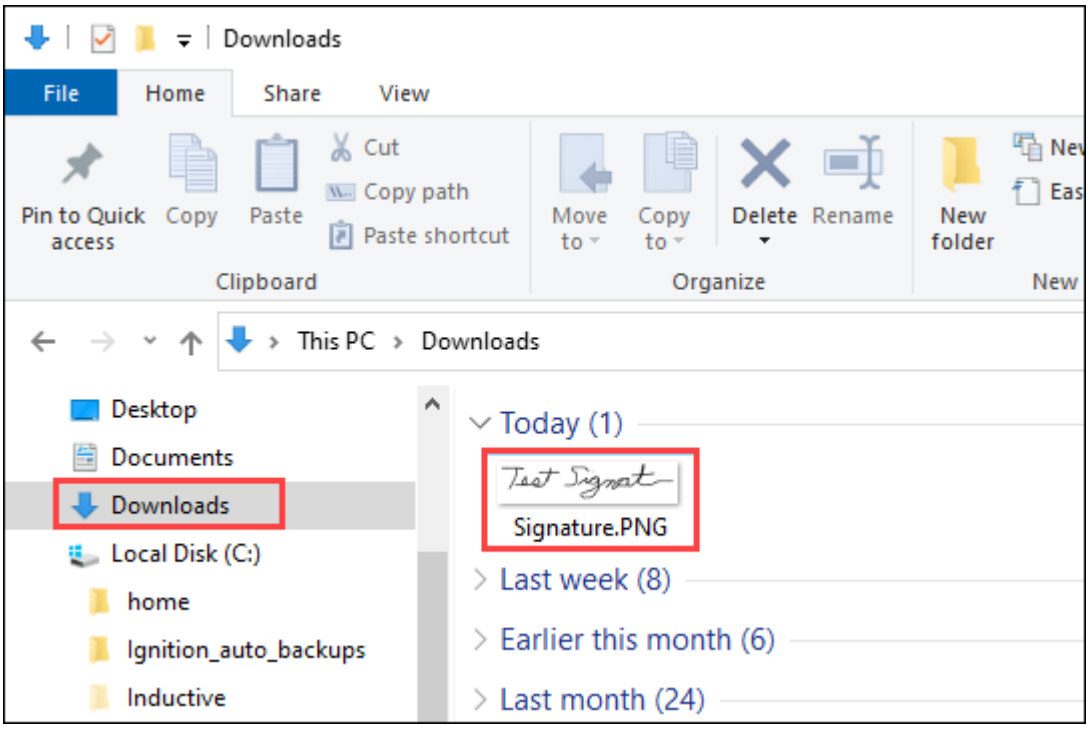
7. Save your project.

8. Open a Perspective Session with the view that has the Signature Pad component.

9. Sign the component and click Submit.



10. An image file is saved to your computer. In this example, we're running Ignition on Windows. The file Signature.PNG appears in our Downloads folder:


Downloads
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New
This PC
Downloads
Desktop
Documents
Downloads
Local Disk (C:)
home
Ignition_auto_backups
Inductive
Today (1)
Signature.PNG
Last week (8)
Earlier this month (6)
Last month (24)

Perspective - Signature Pad Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Signature Pad** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onSignatureSubmitted

Event is fired after the Gateway has received a submitted signature.

> **NOTE**
>
> This component event is designed to be used in tandem with a run action script. Within the script action, special properties and methods are available on the event object, which is passed to the script action as a parameter.

event.signature

- Object Path
  - event.signature
- Type
  - String
- Description
  - Base64-encoded PNG **DataURL** of the submitted signature.

event.signatureFile.name

- Object Path
  - event.signatureFile.name
- Type
  - String
- Description
  - A name for the signature file.

event.signatureFile.size

- Object Path
  - event.signatureFile.size
- Type
  - Integer
- Description
  - The size of the signature image file in bytes.

event.signatureFile.copyTo(filePath)

- Object Path
  - event.signatureFile.copyTo()
- Description
  - Saves the uploaded signature file at a location accessible to the Gateway.
- Parameters
  - String **filePath** - The path to where the file should be saved on the Gateway.
- Return
  - None

event.signatureFile.getBytes()

- Object Path
  - event.signatureFile.getBytes()
- Description
  - Returns a bytearray of the image, allowing the signature file to be saved from the session (with system.perspective.download()).
- Parameters
  - None
- Return
  - byteArray - The raw data of the incoming file.

event.signatureFile.getString()

- Object Path
  - event.file.getString()
- Description
  - Fetches the incoming file data and attempts to parse it as a string via UTF-8 (Eight-bit UCS Transformation Format) encoding. Defaults to UTF-8 (super common), but can use other character sets. Passed as a string, for example getString("UTF_16BE).
- Parameters
  - None
- Return
  - byteArray - The raw data of the incoming signature file.

onSignatureCleared

This event is fired when the Gateway has received a signal that the signature has been cleared.

> **NOTE**

> This component event is designed to be used in tandem with a run action script. Within the script action, special properties and methods are available on the **event** object, which is passed to the script action as a parameter.

event

- Object Path
  - event
- Type
  - Null
- Description
  - An empty event object.

Component Functions

.clearSignature()

- Description
  - Clears the current signature on the component
- Parameters
  - None
- Return
  - Nothing

.submitSignature()

- Description
  - Submits the signature, triggering the onSignatureSubmitted component event.
- Parameters
  - None

- Return
    - Nothing

Extension Functions

This component does not have extension functions associated with it.

Perspective - Slider

Component Palette Icon:



Description

The Slider component lets the user drag an indicator along a scale to choose a value in a range. Enable the "show" and "interval" properties under "labels" to visually display the values within a range. The slider can be configured to orient horizontally or vertically with the "orientation" property.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
value	Value represented by slider handle. Current value of the slider.	value: numeric
min	The minimum value for the slider scale: all the way left or down.	value: numeric
max	The maximum value for the slider scale: all the way right or up.	value: numeric
orientation	Specifies whether the slider track is aligned vertically or horizontally.	value: boolean
step	Intervals along track at which a value may be set. Specifies the size of increments between values of the slider. **Note:** This does not force the value into that step size. Setting the slider value manually or through a binding will cause it to show the actual value.	value: numeric
labels	Label settings along the track. • **show**: If true, displays labels at periodic values along track. Value is boolean. • **interval**: Interval at which to display periodic labels along track. Value is numeric.	object
enabled	Whether slider interaction is currently active. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a **script action** to the onClick event, the script will fire when the user clicks on the Slider.	value: boolean
handleColor	Color of slider handle. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector Reference**.	color
trackColor	Color of slider track. See **Color Selector Reference**.	color

Name	Description	Property Type
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

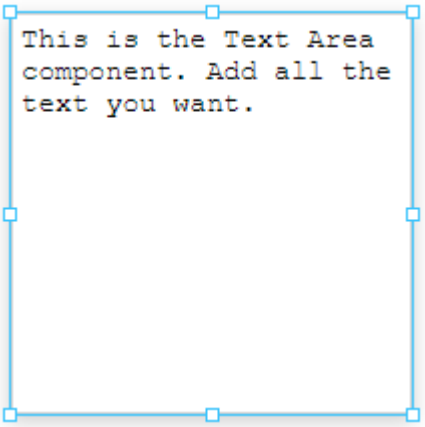
The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

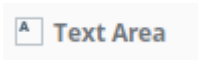


Property	Value
props.value	65
props.orientation	vertical
props.step	5
props.labels.show	true
props.labels.interval	10
props.handleColor	#8AFF8A
props.trackColor	#CCFFFF

Perspective - Text Area



Component Palette Icon:



Description

Suitable for multi-line text display and editing. Will scroll vertically on demand. Horizontal scroll is determined by the "wrap" property.

Properties

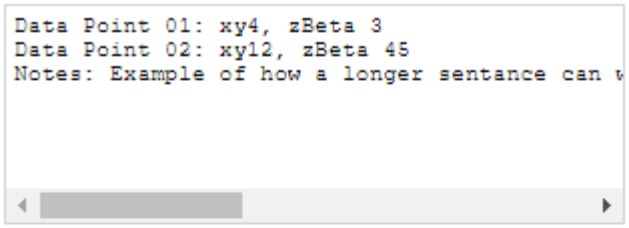
Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
text	Text to display.	value: string
placeholder	Text displayed when Text Area is empty.	value: string
enabled	If true, user is allowed to alter text.	value: boolean
deferUpdates	When true, updates to props.text will be deferred until focus is lost or enter is pressed.	value: boolean
rejectUpdatesWhileFocused	When true, props.text will not accept updates from external sources while focused.	value: boolean
resize	Sets whether text is resizable, and if so, in which direction: none, both, horizontal, or vertical.	value: string
wrap	Specifies how to wrap text: hard, soft, or off. (Soft wrap is the break resulting from a line wrap or word wrap. Hard wrap is an intentional break, which moves text to the next line, or creates a new paragraph).	value: string
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
spellcheck	When true, text containing potential spelling errors will be underlined in red while the text is being edited in a launched session. Be aware there may be some slight behavioral differences in spellcheck error detection depending on the session browser.	value: boolean

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

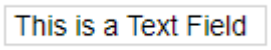




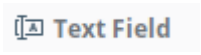
This example shows the Text Area in two states: the top image is with text entered and the bottom image is without text entered.

Property	Value
props.text	Data Point 01: xy4, zBeta 3 Data Point 02: xy12, zBeta 45 Notes: Example of how a longer sentence can wrap (or not) around to the next line. Enter was pressed between each of these three lines.
props.placeholder	Lab Data Entry
props.wrap	off

Perspective - Text Field



Component Palette Icon:



Description

The Text Field component is used for input of any single-line text. This component will accept any alpha-numeric input.

If you need a field that accepts multiple lines of text, see the **Perspective - Text Area** component. If you're looking for a numeric field, see the **Perspective - Numeric Entry Field** component.

User Interaction

The Text Field component properties have impact on the way a user can interact with a table in the runtime.

Interaction	General
Enabled	When the enabled property is set to true, a user can edit the text field in the runtime. The user must double click on the field or press enter in order to edit the field. When done, press enter or leave the field, and the text field becomes non-editable again. When the `enabled` property is set to false, it is not editable even when it receives input focus.
General	The Text Field also supports the reject updates during edit feature. This feature ignores updates coming from property bindings while the component is being edited by a user.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

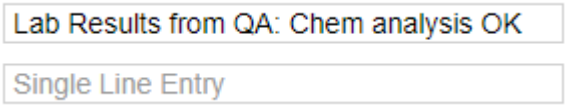
Name	Description	Property Type
text	Text to display.	value: string
placeholder	Text displayed when Text Field is empty.	value: string
enabled	If true, user is allowed to alter text.	value: boolean
deferUpdates	When true, updates to props.text will be deferred until focus is lost or enter is pressed.	value: boolean
rejectUpdatesWhileFocused	When true, props.text will not accept updates from external sources while focused.	value: boolean
spellcheck	When true, text containing potential spelling errors will be underlined in red while the text is being edited in a launched session. Be aware there may be some slight behavioral differences in spellcheck error detection depending on the session browser.	value: boolean
styles	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The

Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example



This example shows the Text Field in two states: the top image is with text entered and the bottom image is without text entered.

Property	Value
props.text	Lab Results from QA: Chem Analysis OK
props.placeholder	Single Line Entry

Perspective - Toggle Switch



Component Palette Icon:



Description

The Toggle Switch represents a bit: on (selected) or off (not selected). By default, when the switch is selected the color is blue. It is gray when it is not selected. Logically, this component is very similar to the Checkbox component.

The Toggle Switch component has three pre-configured **variants**:

- No Text: Default layout with no text.
- Text Right: Layout with text on the right of the Toggle Switch.
- Text Left: Layout with text on the left of the Toggle Switch.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
selected	The selected state of the Toggle Switch.	value: boolean
label	Settings for the label for the Toggle Switch. • **text**: Text for the Toggle Switch. Text can also be entered directly by deep selecting the Toggle Switch component, which enables inline editing. Changes are immediately reflected in the text property field. Value is string. • **position**: Text position relative to the Toggle Switch: right or left. Value is boolean. • **style**: Modify text style using the style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
color	Color settings for the Toggle Switch for each state it is in. Colors can be chosen from a color wheel, color palette, or entered as RGB or HSL values. See the **Color Selector** page for more information. • **selected**: Color of the Toggle Switch when selected (on). • **unselected**: Color of the Toggle Switch when unselected (off). • **New in 8.3.3** **disabled**: Color of the Toggle Switch when disabled by the `enabled` property. ◦ **selected**: Color of the Toggle Switch when selected (on) and disabled. ◦ **unselected**: Color of the Toggle Switch when unselected (off) and disabled.	object
enabled	Whether the user should be allowed to alter the Toggle Switch's selected state. Default is true. **Note:** If the component is disabled, scripts can still run on the component. For example, if you add a script action to	value: boolean

Name	Description	Property Type
	the **onClick** event, the script will fire when the user clicks on the Toggle Switch.	
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

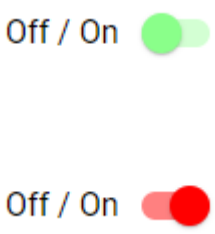


Image above shows both selected and deselected positions for the following properties.

Property	Value
props.label.text	Off / On
props.label.position	left
props.color.selected	#FF0000
props.color.unselected	#8AFF8A

Perspective - Navigation Palette

One of the most important aspects to consider when developing a Perspective application is a solid navigation design so the user knows where they are, where they've been, and where they are going.

The following navigational components provide you with design strategy options to navigate within a Perspective Session, a link pointing to a page containing the component's description, properties, and an example of how to configure it.

Horizontal Menu

Link

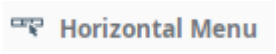
The Link component allows users to create a hyperlink that points to a destination such as a page, view, resource…

Menu Tree

Perspective - Horizontal Menu



Component Palette Icon:



Description

Horizontal Menu component enables you to build a menu structure by setting up multiple links to different page URLs from the component. The Horizontal Menu occupies a large amount of horizontal space and a comparatively small amount of vertical space. The menu starts with a list of root-level menu items that make up the main display area of the component.

If you have more menu items than will fit the width of the component, arrow buttons appear to enable you to scroll left and right through all menu items. The control can also be disabled as a whole.

Each menu item can be configured with a target that will serve as either a link to a page that should be shown (e.g. “/my-page”), or a link to an external web page (e.g. “**http://www.inductiveautomation.com/**”). They can also be given a list of child menu items that will show temporarily in a popup as the user is interacting with them. Additionally, menu items can be labeled, disabled, and be given an icon to show to the left of their label.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type

Name	Description	Property Type
items	Configure items representing the main menu items. • **enabled**: Whether this option is currently enabled to perform its action or render its submenu. Value is boolean. • **target**: A URL (**http://www.inductiveautomation.com/**) or mounted path to a page (/my-page). If "items" is empty (no subtree to this item), this will navigate to that location. Value is string. • **items**: Configure items representing child menu items from this option. If defined, a submenu will branch from here with these options. ◦ **enabled**: Whether this option is currently enabled to perform its action or render its submenu. Value is boolean. ◦ **target**: A URL (external) or mounted path to a page. If "items" is empty (no subtree to this item), this will navigate to that location. Value is string. ◦ **items**: Configure items representing child menu items from this option. If defined, a submenu will branch from here with these options. Value is array. ◦ **icon**: Icon image appended to the left of the menu item, with two child properties: **path** and **color**. The **path** property creates a shorthand path to icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. The **color** property sets the color of the icon. Can also assign the "fill" property in styles. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**. ◦ **label**: Text to display for this menu item label. Value is string. ◦ **style**: Sets a style for this item. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	

Name	Description	Property Type
enabled	If true, the component is enabled/active. Default is true.	value: boolean
itemStyle	Sets a style for all the items in the component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
style	Sets a style for the entire component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Scripting

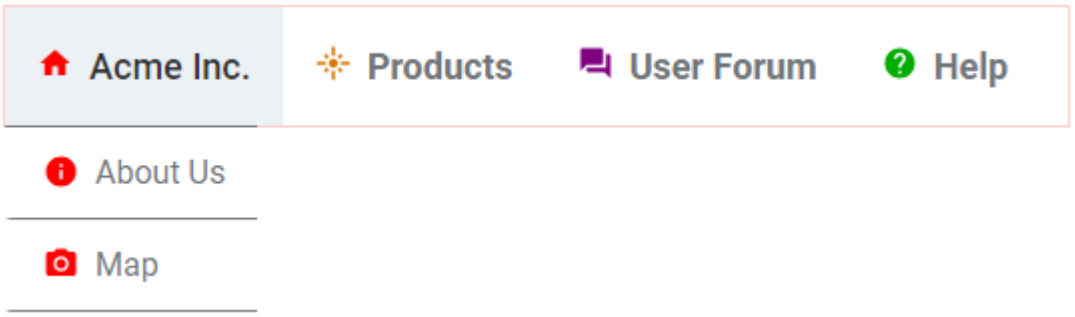
See the **Perspective - Horizontal Menu Scripting** page for the full list of scripting functions available for this component.

Examples

In addition to the examples below, learn more about the Horizontal Menu component on the **Navigating with the Horizontal Menu Component** page.

Example 1

This example shows a Horizontal Menu with four items. The first item also has two subitems. Each item links to a webpage. Icons are taken from the Material Design icons that can be found here: **https://material.io/tools/icons/**.



Top Level Properties

Property	Value
props.items.0.enabled	true
props.items.0.target	https:// (link to Acme home)
props.items.0.icon.path	material/home
props.items.0.icon.color	#FF0000
props.items.0.label	Acme Inc.
props.items.1.enabled	true
props.items.1.target	https:// (link to products)
props.items.1.icon	material/flare
props.items.1.color	#D97700
props.items.1.label	Products
props.items.2.enabled	true
props.items.2.target	**https://** (link to forum)
props.items.2.icon.path	material/forum
props.items.2.icon.color	#800080
props.items.2.label	User Forum
props.items.3.enabled	true
props.items.3.target	**https://** (link to help)
props.items.3.icon.path	material/help
props.items.3.icon.color	#00AC00
props.items.3.label	Help

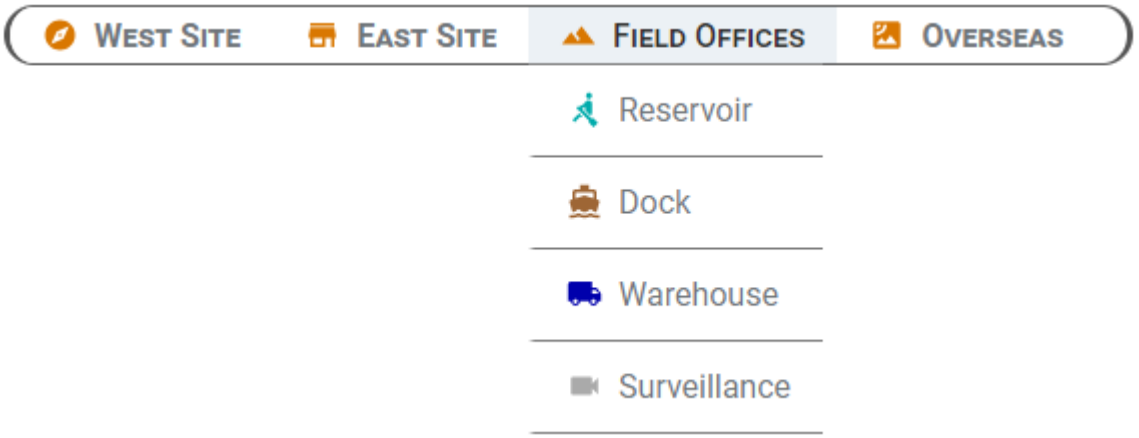
Property	Value
props.style.borderStyle	solid
props.style.color	#FF4747
props.style.fontSize	18px
props.style.fontWeight	bold
props.style.borderWidth	1pc
props.style.borderColor	#FFCCCC

Sub Level Properties of Item 0

Property	Value
props.items.0.items.0.target	**https://** (link to about)
props.items.0.items.0.icon.path	material/info
props.items.0.items.0.icon.color	#FF0000
props.items.0.items.0.label	About Us
props.items.0.items.0.target	/screen_2
props.items.0.items.0.enabled	true
props.items.0.items.1.target	**https://** (link to about)
props.items.0.items.1.icon.path	material/local_see
props.items.0.items.1.icon.color	#FF0000
props.items.0.items.1.label	Map
props.items.0.items.1.target	**https://** (link to map)
props.items.0.items.1.enabled	true

Example 2

This example shows a Horizontal Menu with four items that are linked to views within the Perspective project. The third item in the list, Field Offices, has four subitems. Each item links to a view for a different field office. Icons are taken from the Material Design icons that can be found here: **https://material.io/tools/icons/**.



Top Level Properties

Property	Value
props.items.0.enabled	true
props.items.0.target	/west
props.items.0.icon.path	material/explore
props.items.0.icon.color	#D97700
props.items.0.label	West Site
props.items.1.enabled	true
props.items.1.target	/View_East
props.items.1.icon	material/store
props.items.1.color	#D97700
props.items.1.label	East Site
props.items.2.enabled	true
props.items.2.icon.path	material/landscape
props.items.2.icon.color	#800080
props.items.2.label	Field Offices
props.items.3.enabled	true
props.items.3.target	/satellite
props.items.3.icon.path	material/satellite
props.items.3.icon.color	#00AC00
props.items.3.label	Overseas
props.style.borderStyle	solid

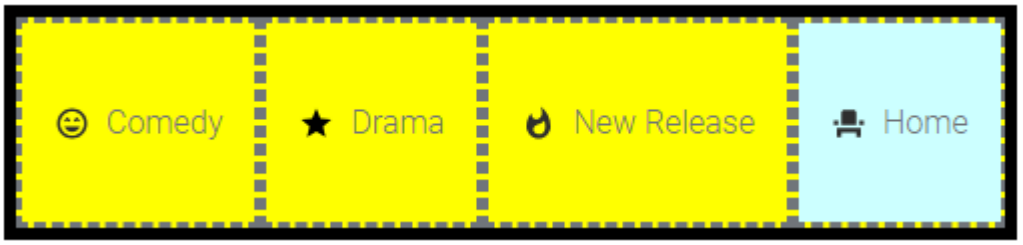
Property	Value
props.style.fontStyle	normal
props.style.fontVariant	small-caps
props.style.borderRadius	16
props.style.borderColor	#555555

Sub Level Properties of Item 2

Property	Value
props.items.2.items.0.enabled	true
props.items.2.items.0.target	/reservoir
props.items.2.items.0.icon.path	material/rowing
props.items.2.items.0.icon.color	#00ACAC
props.items.2.items.0.label	Reservoir
props.items.2.items.1.enabled	true
props.items.2.items.1.target	/dock
props.items.2.items.1.icon.path	material/directions_boat
props.items.2.items.1.icon.color	#9E6635
props.items.2.items.1.label	Dock
props.items.2.items.2.enabled	true
props.items.2.items.2.target	/warehouse
props.items.2.items.2.icon.path	material/local_shipping
props.items.2.items.2.icon.color	#0000AC
props.items.2.items.2.label	Warehouse
props.items.2.items.2.enabled	true
props.items.2.items.2.target	/surveillance
props.items.2.items.2.icon.path	material/videocam
props.items.2.items.2.icon.color	#AAAAAA
props.items.2.items.2.label	Warehouse

Example 3

This example shows a Horizontal Menu with four items.



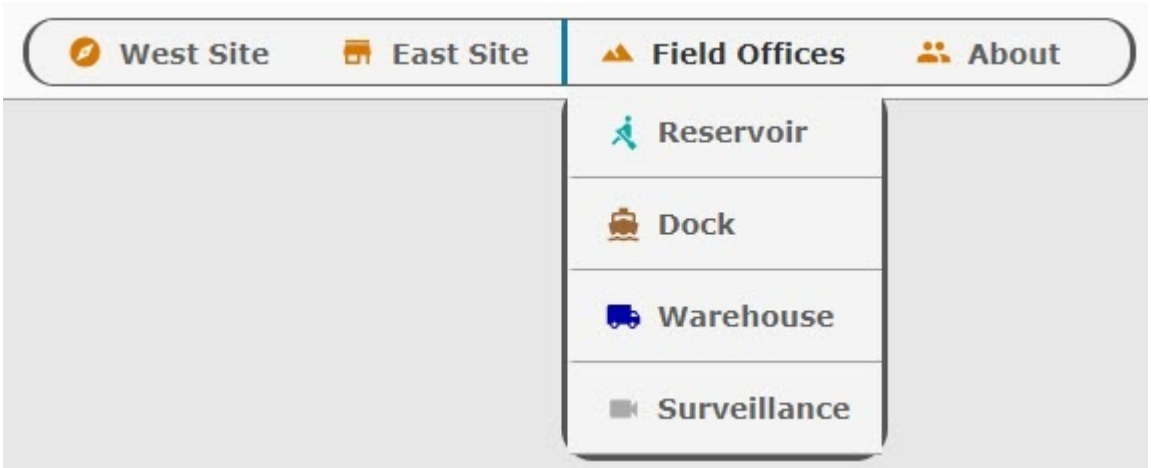
Top Level Properties

Property	Value
props.items.0.enabled	true
props.items.0.icon.path	material/sentiment_very_satisfied
props.items.0.icon.color	#D97700
props.items.0.label	Comedy
props.items.1.enabled	true
props.items.1.icon	material/star
props.items.1.label	Drama
props.items.2.enabled	true
props.items.2.icon.path	material/whatshot
props.items.2.label	New Release
props.items.3.enabled	true
props.items.3.icon.path	material/event_seat
props.items.3.label	Home
props.items.3.style.backgroundColor	#CCFFFF
props.itemStyle.backgroundColor	#FFFF00
props.itemStyle.borderStyle	dashed
props.style.borderStyle	solid
props.style.borderWidth	6px
props.style.borderColor	#000000

Navigating with the Horizontal Menu Component

The Horizontal Menu component enables you to build a menu structure by setting up multiple links to different page URLs from the component. Our example has a menu with links to three internal pages and one external page on the Internet.

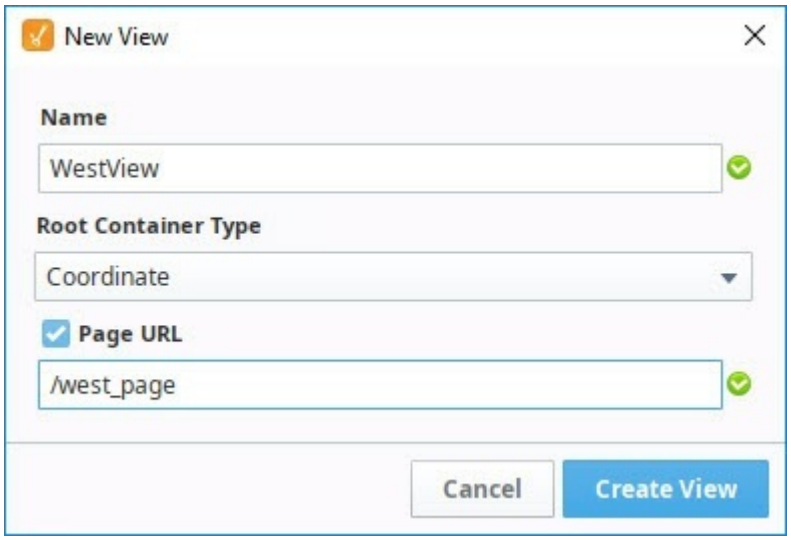
This example shows a Horizontal Menu with four items. Three items are linked to views within the Perspective project and the fourth is linked to a website. The third item in the list, Field Offices, has four subitems.



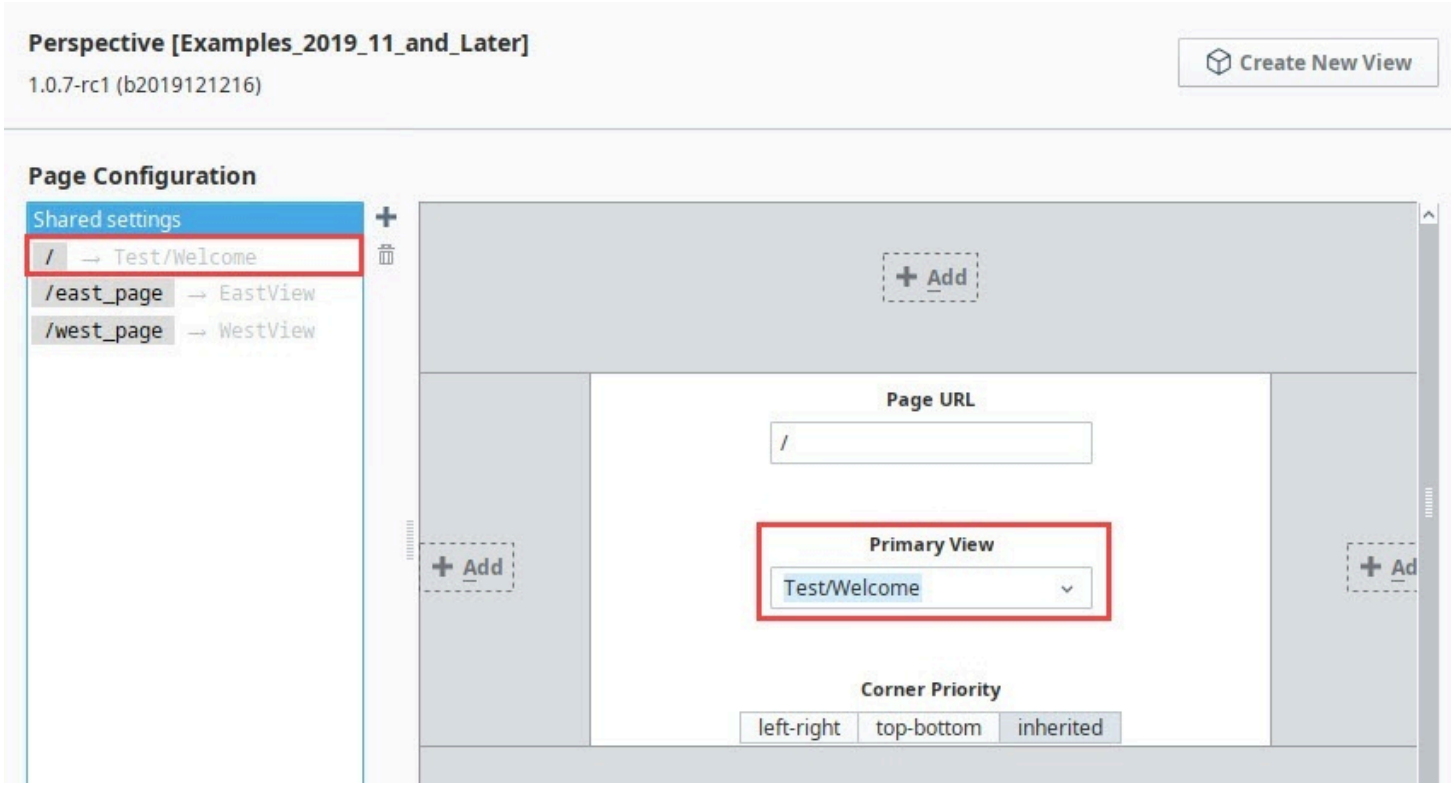
Initial Project Setup

To begin with we have created three views: **WestView**, **EastView**, and **Welcome**. They are each coordinate type views. As we create each new view, we checked the Page URL property and added a page name.

View Name	Page URL
WestView	/west_page
EastView	/east_page
Welcome	/



On Page Configuration, you'll note that each page is already set up with a Primary view.



Now we're ready to start building the navigation.

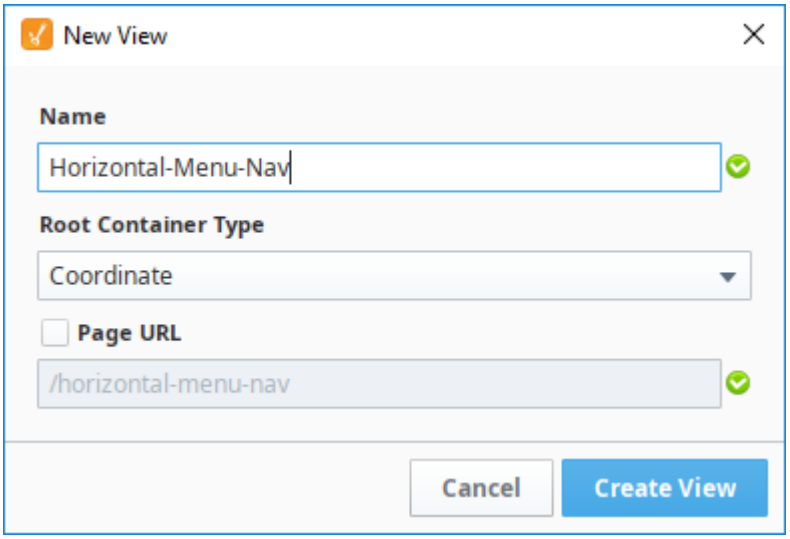
Set Up a Header View

The first thing we'll set up is a view that will hold the Horizontal Menu component. We'll use this view as a header for our pages within this project.

1. In the Project Browser, right click on Views and select the **NewFolder** option. Name the folder "Header".

2. Right click on the Header folder and select the **NewView** option.

- Name: **Horizontal-Menu-Nav**
- Layout: **Coordinate**
- Page URL: unchecked



3. Click **Create View**.
4. In the Property Editor for the view, set the **defaultSize** property as follows:
   - width: **800**
   - height: **50**
5. Drag a Horizontal Menu component onto the view.
6. In the Property Editor, set the Position Properties as follows:

Property	Value
position.x	15
position.y	10
position.width	550
position.height	30

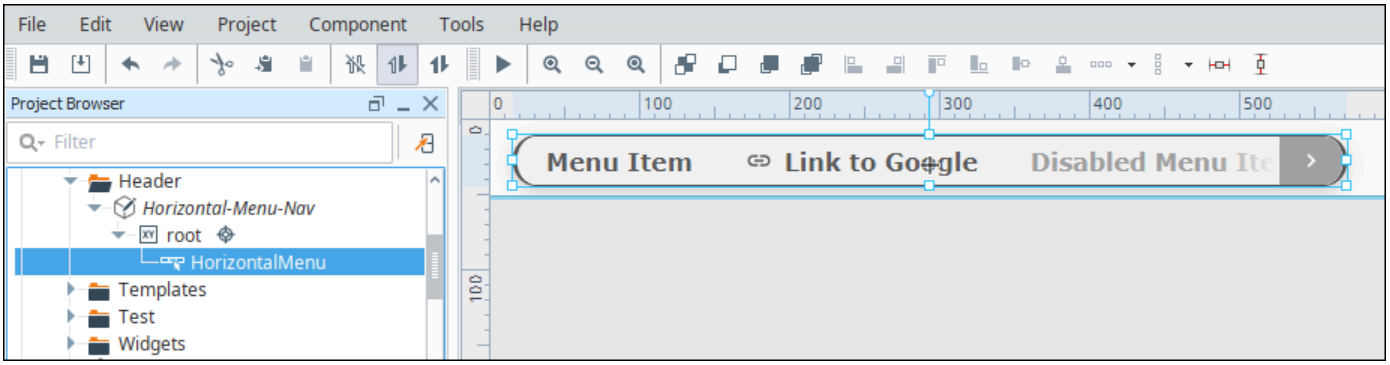
7. In the Property Editor, scroll down to style and click the **ModifyStyle** icon.
   i. Expand the Text section and set the style options as follows:

Property	Value
props.style.fontWeight	bolder
props.style.fontFamily	Verdana
props.style.fontSize	14px

ii. Expand the Border section and set the style options as follows:

Property	Value
props.style.borderStyle	solid
props.style.borderColor	#555555
props.style.borderRadius	16px
All Corners	(selected)

Your Designer will look like this at this point:

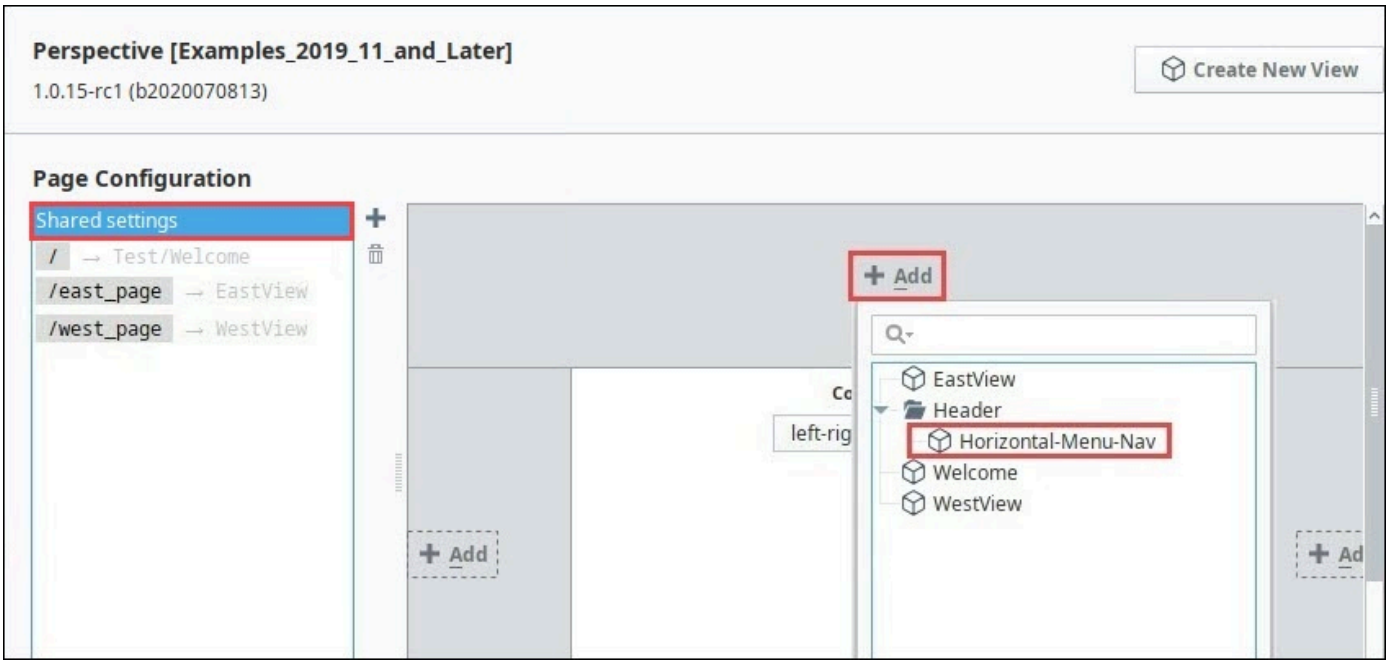


8. Open Page Configuration by clicking on the **Settings** ⚙ icon at the bottom left of the Designer window.

9. Under Page Configuration, click on **Shared Settings**.

10. In the header part of the page mockup, click on the **Add** ✚ icon.

11. Select the Horizontal-Menu-Nav view from the dropdown.



12. Click **OK**. The Horizontal-Menu-Nav view will now appear at the top of all pages in the project.

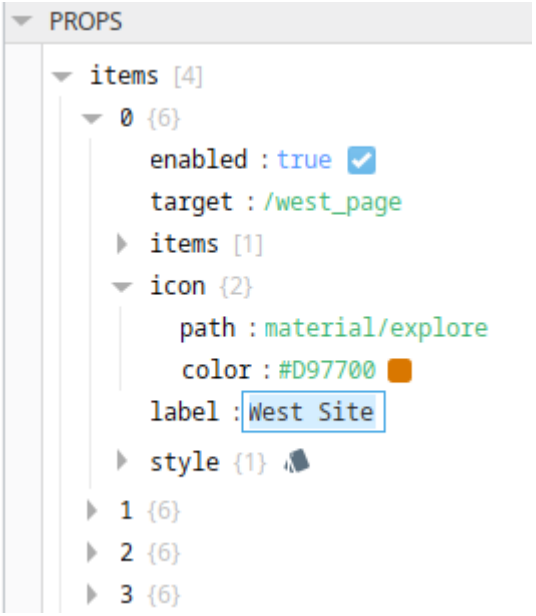
Configure the Tabs in the Horizontal Menu

Now let's set the properties for each of the four tabs in the Horizontal Menu. Each tab will have a display a name and an icon and will have a target page or website to open when clicked.

1. In the Property Editor, expand the items property and set the following for item 0:

Property	Value
props.items.0.enabled	true
props.items.0.target	/west_page
props.items.0.icon.path	material/explore
props.items.0.icon.color	#D97700
props.items.0.label	West Site

The Property Editor will look like this:



2. In the Property Editor, set the following for item 1:

Property	Value
props.items.1.enabled	true
props.items.1.target	/east_page
props.items.1.icon	material/store
props.items.1.color	#D97700
props.items.1.label	East Site

3. In the Property Editor, set the following for item 2:

> **NOTE**
>
> Do not set a props.items.2.target property value for this tab because we will set up dropdown tabs in the next section.

Property	Value
props.items.2.enabled	true
props.items.2.icon.path	material/landscape
props.items.2.icon.color	#D97700
props.items.2.label	Field Offices

4. In the Property Editor, set the following for item 3:

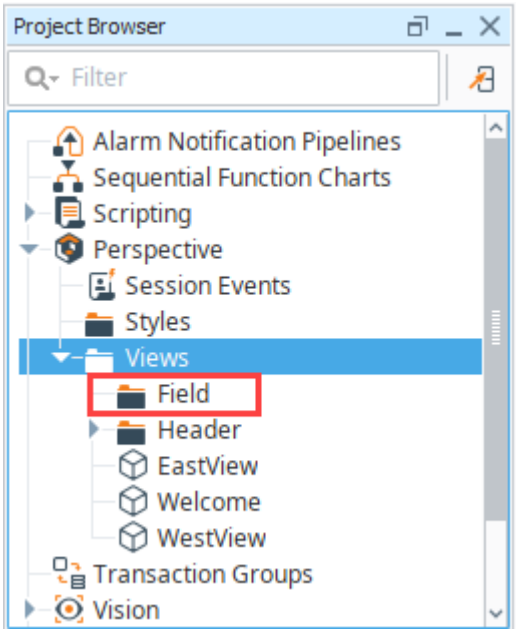
> **NOTE**
>
> This tab uses a website as its target, therefore it does not need to target a page within Perspective.

Property	Value
props.items.3.enabled	true
props.items.3.target	http://inductiveautomation.com/about/
props.items.3.icon.path	material/people
props.items.3.icon.color	#D97700
props.items.3.label	About

Configure the Field Offices Tab in the Horizontal Menu

The third tab in the Horizontal Menu is titled "Field Offices." Instead of navigating to one page, this tab has a dropdown menu with four options on it: Reservoir, Dock, Warehouse, and Surveillance.

1. To start, we created a new folder in Views called **Field**.

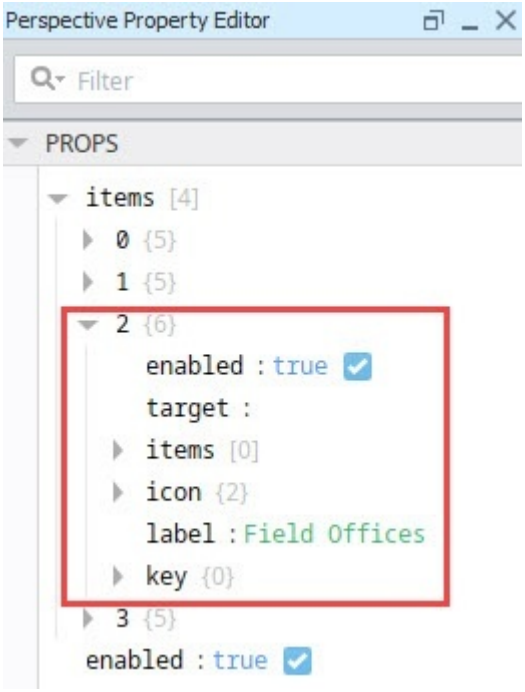


2. Within the Field folder, we create four views: **Reservoir**, **Dock**, **Warehouse**, and **Surveillance**. Make them Coordinate types and set up a page URL for each.

View Name	Page URL
Reservoir	/reservoir_page
Dock	/dock_page
Warehouse	/warehouse_page
Surveillance	/surveillance_page

3. Open the Horizontal-Menu-Nav view and select the Horizontal Menu component.

4. In the Property Editor, expand the properties for Item 2.



5. Select the props.items.2.items property.

6. Click the Add Add Array Element... icon to add four items.

7. Set the properties for Item 0 as follows:

Property	Value
props.items.2.items.0.enabled	true
props.items.2.items.0.target	/reservoir_page
props.items.2.items.0.icon.path	material/rowing
props.items.2.items.0.icon.color	#00ACAC
props.items.2.items.0.label	Reservoir

The Property Editor will look like this:



8. Now we'll do the same for the other three items. Set the properties for Item 1 as follows:

Property	Value
props.items.2.items.1.enabled	true
props.items.2.items.1.target	/dock_page
props.items.2.items.1.icon.path	material/directions_boat
props.items.2.items.1.icon.color	#9E6635
props.items.2.items.1.label	Dock

9. Set the properties for Item 2 as follows:

Property	Value
props.items.2.items.2.enabled	true
props.items.2.items.2.target	/warehouse_page
props.items.2.items.2.icon.path	material/local_shipping
props.items.2.items.2.icon.color	#0000AC
props.items.2.items.2.label	Warehouse

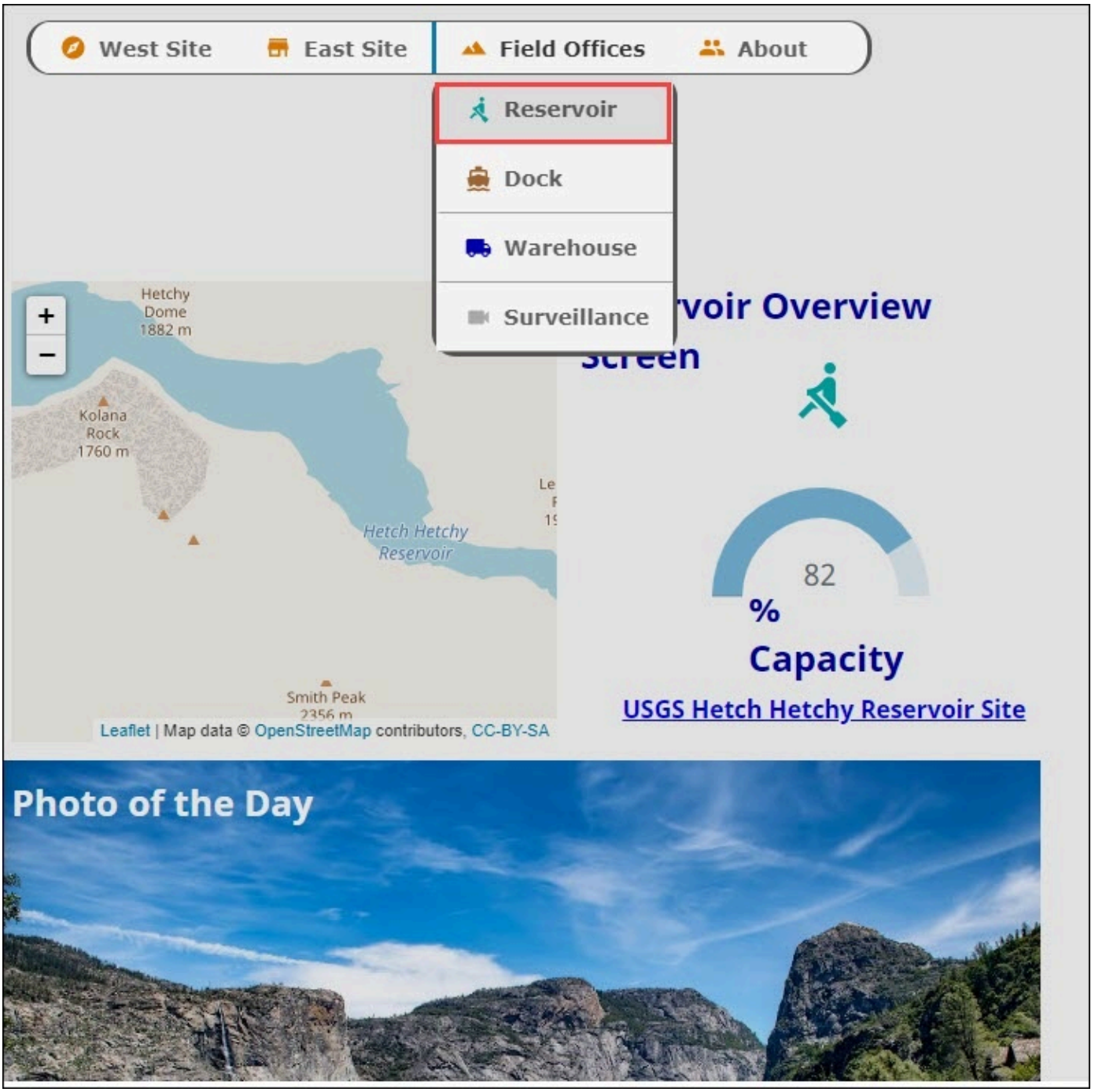
10. Set the properties for item 3 as follows:

Property	Value
props.items.2.items.3.enabled	true
props.items.2.items.3.target	/surveillance_page
props.items.2.items.3.icon.path	material/videocam
props.items.2.items.3.icon.color	#AAAAAA
props.items.2.items.3.label	Surveillance

11. Save your project.

12. Click **Tools** > **Launch Perspective** > **Launch Session**.

13. Click on the tabs in the header to view different pages. For our example, we have put a few components on each view. Here is an example of what the Field Offices > Reservoir page might look like:


West Site
East Site
Field Offices
About
Reservoir
Dock
Warehouse
Surveillance
Hetchy Dome 1882 m
Kolana Rock 1760 m
Hetch Hetchy Reservoir
Smith Peak
Leaflet | Map data © OpenStreetMap contributors, CC-BY-SA
rvoir Overview
Screen
82
%
Capacity
USGS Hetch Hetchy Reservoir Site
Photo of the Day

Perspective - Horizontal Menu Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Horizontal Menu** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onItemClicked

Fired when an item is selected.

event.enabled

- Object Path
  - event.enabled
- Type
  - Boolean
- Description
  - Whether the item interacted with is enabled.

event.label

- Object Path
  - event.label
- Type

    - String
- Description
    - Text to display for this option.

event.path

- Object Path
    - event.label
- Type
    - List
- Description
    - A list containing the item indexes leading to the item that was clicked.

event.target

- Object Path
    - event.target
- Type
    - String
- Description
    - A URL (external) or a mounted path to a page.

Component Functions

This component does not have component functions associated with it.

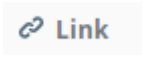
Extension Functions

This component does not have extension functions associated with it.

Perspective - Link



Component Palette Icon:



Description

The Link component allows users to create a hyperlink that points to a destination such as a page, view, resource, or mount path that they can quickly navigate to. Links are easily identifiable because they typically have a different color font then the rest of the content, and an underscore when you hover over them. To configure a link, enter the URL for the destination, and the name of the link in the 'text' property.

Properties

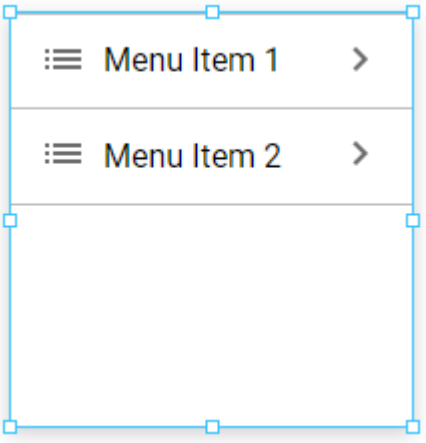
Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
url	A URL, URL fragment, or a Page URL given to a Page. If the target is a Page, its Page URL can be found in the Page Configuration settings of the project. See **Page URLs**.	value: string
text	Text to display in the link.	value: string
target	Specifies where to display the linked URL. Options are: • **self** for the current tab/context • **tab** or 'blank' for a new tab • **parent** for the parent frame tab/context • **top** for the full body of the window Otherwise, supports standard w3c values for anchor link target attributes.	value: string
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Perspective - Menu Tree



Component Palette Icon:



Description

The Menu Tree component can be configured to allow users to navigate pages in a Perspective Session. The Menu Tree defines a hierarchical view of information that can be configured to expand submenu branches and menu items. The subitems can be further expanded to expose more subitems if any exist, and collapsed to hide subitems.

Each menu item has its own path, for example, "Western Region/CA/San Jose" that determines its location in the Menu Tree. The Separation Character property (by default its forward-slash), dictates how the paths are broken up.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
items	Configure items representing nodes in a subtree from this option. If defined, a submenu will branch from here with these options. Click to see the **items** properties.	array
layoutAlignment	Specifies which side of the root menu is aligned to. Submenu slides in from the opposite side.	value: boolean
enabled	If true, this component is currently enabled to perform its actions.	value: boolean
itemStyle	Sets **style options** for the menu tree component. You can also specify a **style class**.	object
headerStyle	Sets **style options** for the menu tree component. You can also specify a **style class**.	object
backActionStyle	Sets **style options** for the menu tree component. You can also specify a **style class**.	object
backActionText	Text to display in prompt to go back to the root menu. You can also specify a **style class**.	value: string
style	Sets a style for this component. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

items Properties

Name	Description	Property Type
target	A url (external), or mounted path to a page. If "items" is empty (no subtree to this item), this will navigate to that location.	value: string
items	Configure items representing nodes in a subtree from this option. If defined, a submenu will branch from here with these options.	array
navIcon	Icon image appended to the right of the menu item. • **path**: Shorthand path to icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string. • **color**: Color of the icon. Can also assign the "fill" property in styles. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	object
label	Menu item label. • **text**: Text to display for this option. Value is string. • **icon**: Image icon added to the right of the label text. ◦ **path**: Shorthand path to icon source, in format: library/IconName. The materials icon library is a the primary source for icons, see **https://fonts.google.com/icons?selected=Material+Icons**. Value is string.	object
visible	Whether this option should be displayed in the menu tree.	value: boolean
enabled	If true, this option is currently enabled to perform its action or render its submenu.	value: boolean

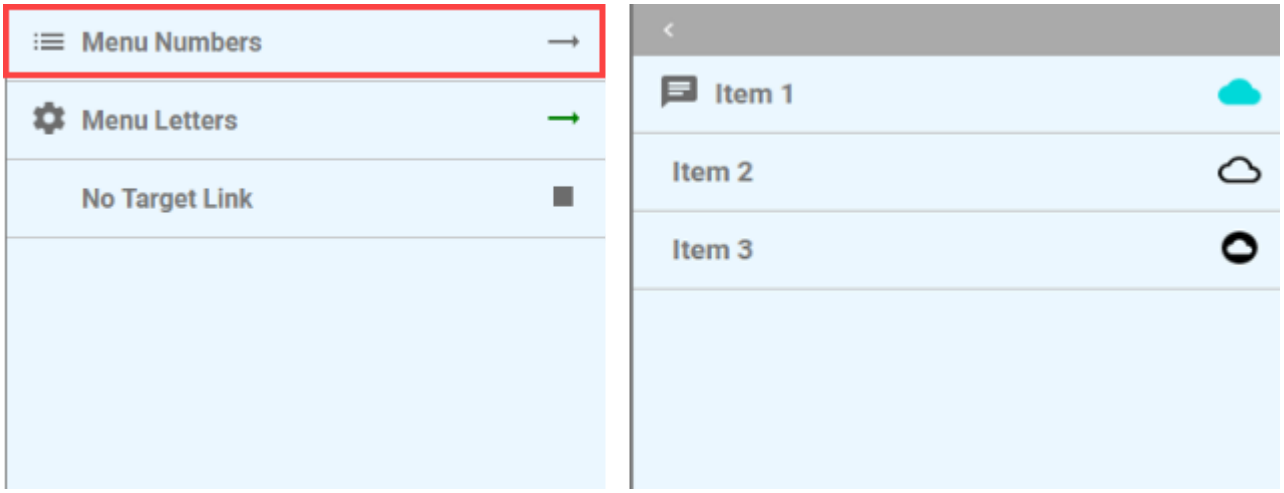
Name	Description	Property Type
showHeader	Whether to display this option's text as a header/title for its submenu.	value: boolean
style	Sets a style for this item. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object
resetOnClick	When true, clicking on the item will cause the Menu Tree to reset back to its root level. Useful in cases where you want to provide your users with a quick way to return to the start of the menus.	value: boolean
backActionText	Text to display in prompt to go back to the previous menu. This configuration property will override the text set to display in the root backActionText property. If left blank, menu items default to the root backActionText.	value: string

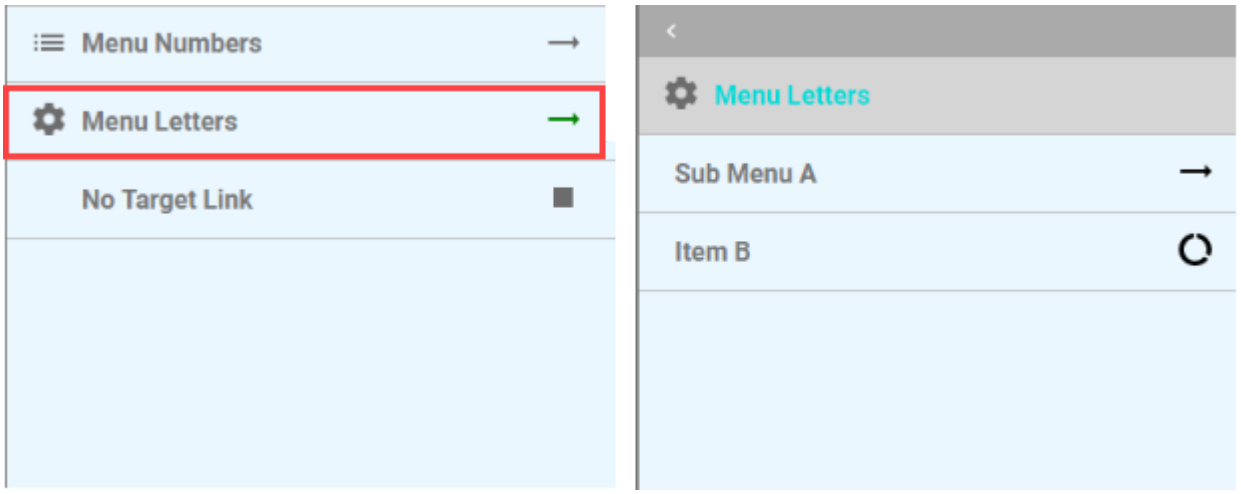
Scripting

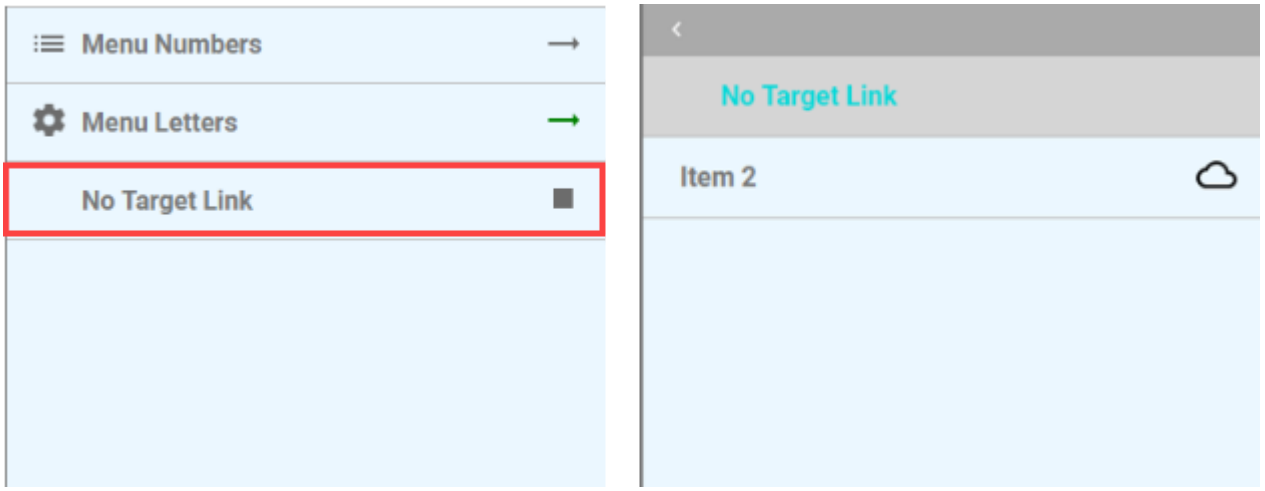
See the **Perspective - Menu Tree Scripting** page for the full list of scripting functions available for this component.

Examples

Example 1 - Basic Menu Tree







You can use links to go to URL links outside your project or internal to your project. For this example assume you have some screens defined in the project page configuration (home gear). Assume the URL's for those are as follows:

- /screen_1

- /screen_2
- /screen_3
- /screen_b
- /screen_a1
- /screen_a2

> **NOTE**
>
> Images on the left side of the Menu Tree are associated with the label.icon.path Images on the right side of the Menu Tree are associated with the navIcon.path target property is the link to the screen to navigate to.

Top Level Properties

Property	Value
props.items.0.navIcon.path	material/trending_flat
props.items.0.navIcon.color	#6C6C6C
props.items.0.label.text	Menu Numbers
props.items.0.label.icon.path	material/list
props.items.0.label.showHeader	false
props.items.1.navIcon.path	material/trending_flat
props.items.1.navIcon.color	#008000
props.items.1.label.text	Menu Letters
props.items.1.label.icon.path	material/settings
props.items.1.label.showHeader	true
props.items.2.navIcon.path	material/stop
props.items.2.navIcon.color	#FF8A8A
props.items.2.label.text	No Target Link
props.items.2.label.icon.path	material/stop_screen_sharing
props.items.2.label.showHeader	true

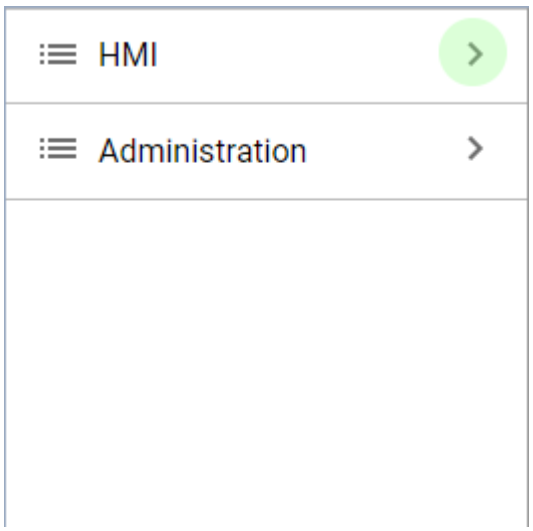
Sub Level Properties of Item 0

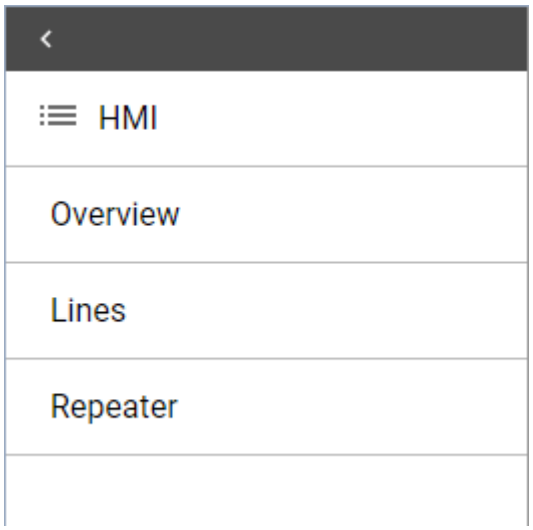
Property	Value
props.items.0.items.0.target	/screen_1
props.items.0.items.0.navIcon.path	material/cloud
props.items.0.items.0.navIcon.color	#00D9D9
props.items.0.items.0.label.text	Item 1
props.items.0.items.0.label.icon.path	material/chat
props.items.0.items.0.label.showHeader	true
props.items.0.items.1.target	/screen_2
props.items.0.items.1.navIcon.path	material/cloud_queue
props.items.0.items.1.label.text	Item 2
props.items.0.items.1.label.showHeader	true
props.items.0.items.2.target	/screen_3
props.items.0.items.2.navIcon.path	material/cloud_circle
props.items.0.items.2.label.text	Item 3
props.items.0.items.2.label.showHeader	true

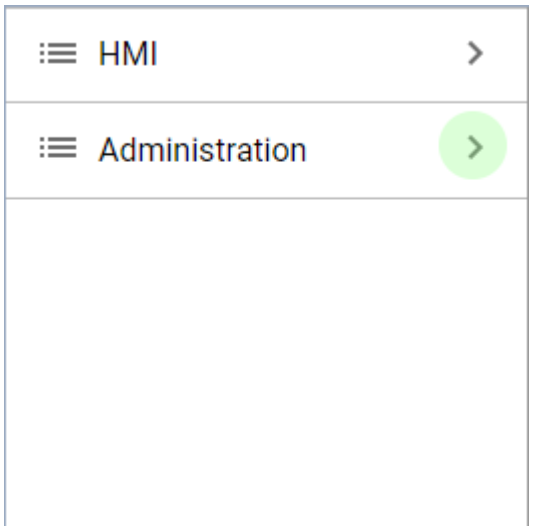
Sublevel Properties of Item 1

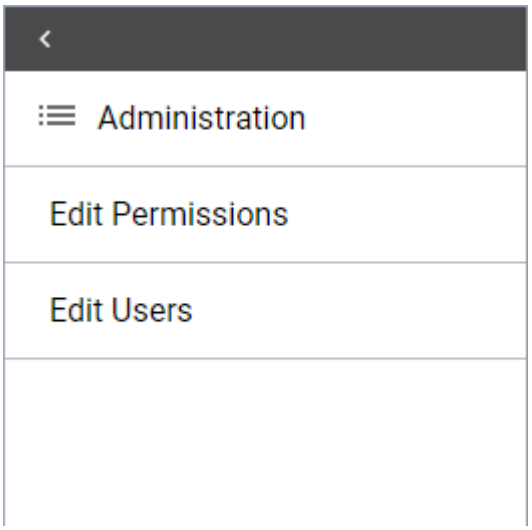
Property	Value
props.items.1.items.0.navIcon.path	material/trending_flat
props.items.1.items.0.label.text	Sub Menu A
props.items.0.items.0.label.showHeader	true
props.items.1.items.1.target	/screen_b
props.items.1.items.1.navIcon.path	material/data_usage
props.items.1.items.1.label.text	Item B
props.items.1.items.0.items.0.target	/screen_a1
props.items.1.items.0.items.0.navIcon.path	material/group
props.items.1.items.0.items.0.navIcon.color	#6C6C6C
props.items.1.items.0.items.0.label.text	Item A 1
props.items.1.items.0.items.0.label.showHeader	true
props.items.1.items.0.items.1.target	/screen_a2
props.items.1.items.0.items.1.navIcon.path	material/group_add
props.items.1.items.0.items.1.navIcon.color	#6C6C6C
props.items.1.items.0.items.1.label.text	Item A 2
props.items.1.items.0.items.1.label.showHeader	true

Example 2 - Tree Navigation









The **items** property for this component is complex. It is an "object" type that has as many levels as you want. Here is an example of the JSON used to create the two level image above. You can copy the text below and paste it into the items property of your Menu Tree component. No code is needed to make this component navigate.

This example assumes several things in order to work:

- Target fields that are blank (HMI and Administration) do not navigate when clicked.
- There are several pages created in the project:
  - /overview
  - /lines
  - /packaging
  - /userPermissions
  - /userEdit
- There are icons stored in ignition.
  - A hamburger menu icon with a path of "menu_list"
  - A greater than symbol with a path of "chevron_right"

items Property

```
[
  {
    "target": "",
    "items": [
      {
        "target": "/overview",
        "items": [],
        "navIcon": {
          "path": ""
        },
        "label": {
          "text": "Overview",
          "icon": {
            "path": ""
          }
        },
        "showHeader": true
      },
      {
        "target": "/lines",
        "items": [],
        "navIcon": {
          "path": ""
        },
        "label": {
          "text": "Lines",
          "icon": {
            "path": ""
          }
        },
        "showHeader": true
      },
      {
        "target": "/packaging",
        "items": [],
        "navIcon": {
          "path": "Packaging"
        },
        "label": {
          "text": "Repeater",
          "icon": {
            "path": ""
          }
        },
        "showHeader": true
      }
    ],
    "navIcon": {
```

```
      "path": "chevron_right",
      "color": "#6C6C6C"
    },
    "label": {
      "text": "HMI",
      "icon": {
        "path": "menu_list"
      }
    },
    "showHeader": true
  },
  {
    "target": "",
    "items": [
      {
        "target": "/userPermissions",
        "items": [],
        "navIcon": {
          "path": ""
        },
        "label": {
          "text": "Edit Permissions",
          "icon": {
            "path": ""
          }
        },
        "showHeader": true
      },
      {
        "target": "/userEdit",
        "items": [],
        "navIcon": {
          "path": ""
        },
        "label": {
          "text": "Edit Users",
          "icon": {
            "path": ""
          }
        },
        "showHeader": true
      }
    ],
    "navIcon": {
      "path": "chevron_right",
      "color": "#6C6C6C"
    },
    "label": {
      "text": "Administration",
```

```
      "icon": {
        "path": "menu_list"
      }
    },
    "showHeader": true
  }
]
```

Perspective - Menu Tree Scripting

This page details the various scripting, component, and extension functions available for **Perspective's Menu Tree** component.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

onItemClicked

Fired when an item is selected.

event.enabled

- Object Path
  - event.enabled
- Type
  - Boolean
- Description
  - Whether the item interacted with is enabled.

event.label

- Object Path
  - event.label
- Type

    - String
- Description
    - Text to display for this option.

event.path

- Object Path
    - event.label
- Type
    - List
- Description
    - A list containing the item indexes leading to the item that was clicked.

event.target

- Object Path
    - event.target
- Type
    - String
- Description
    - A URL (external) or a mounted path to a page.

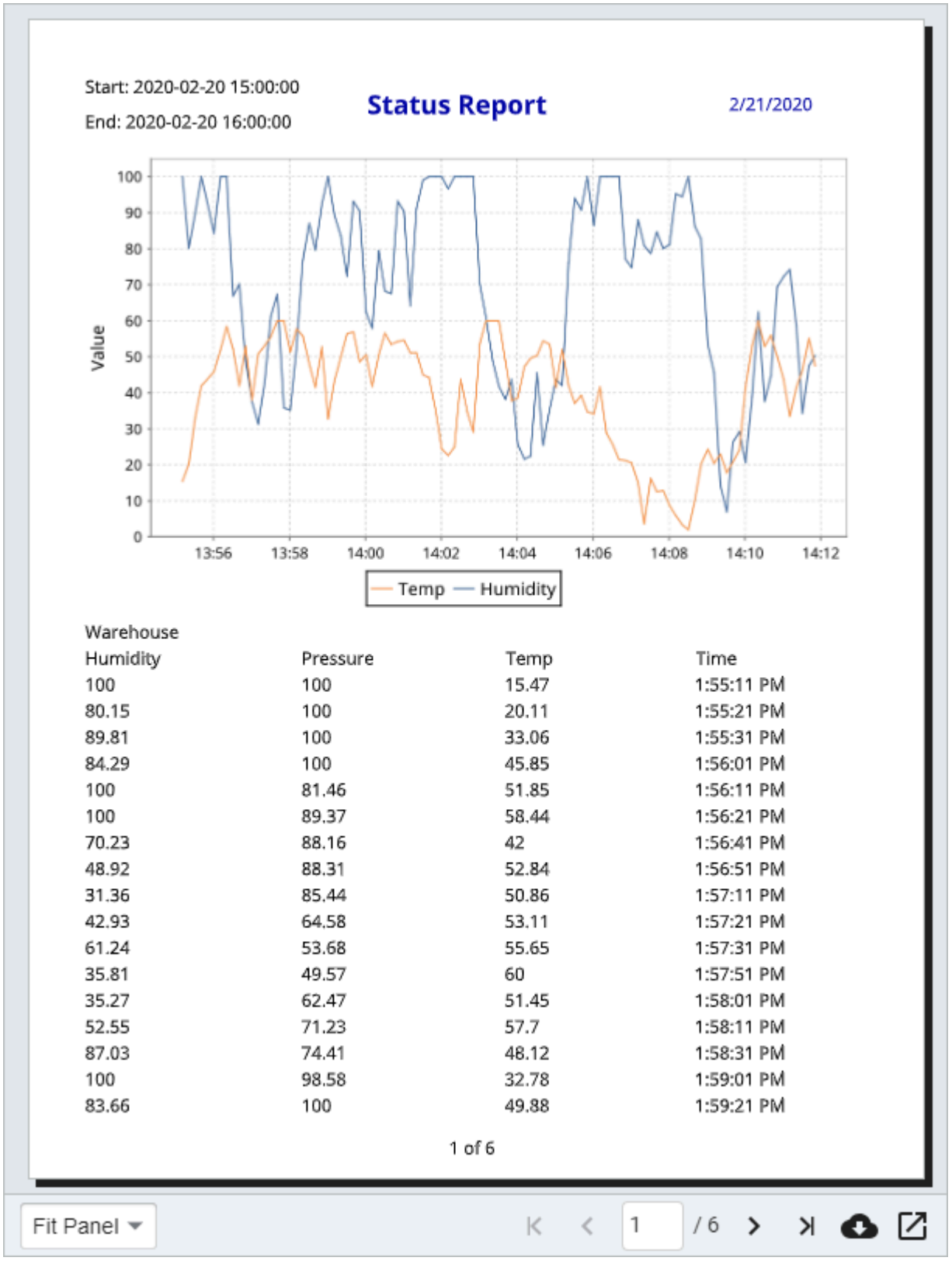
Component Functions

This component does not have component functions associated with it.

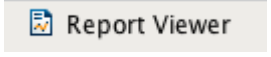
Extension Functions

This component does not have extension functions associated with it.

Perspective - Report Viewer



Component Palette Icon:



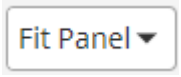
Description

The Report Viewer component allows embedding reports from the Reporting Module into a Perspective views. The Reporting Module must be installed to use the Report Viewer component. To configure the Report Viewer, you must first create a report and provide the Path of that report in the '**source**' property of the component.

You can specify any parameters that you are using in the report as values under the params property, but the parameter names must match. The values specified will be used instead of the default report parameters.

Interface Elements



Icon	Name	Description
	Zoom	Controls the zoom level of the PDF view.
	Pager	Displays the current page and allows the user to increment or decrement pages.
	Download	Downloads the report. After clicking **Download**, the button is disabled until the user either saves the report or cancels the download.
	Popout	Opens the report in a new window.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
source	Path of the report (case sensitive) from the Reporting Module that this component should display. For example, your Report source path might be: "Folder/ReportName". Right-click on the report in the Project Browser to Copy Path.	value: string
params	An object that can be enhanced with **report parameters** to use in a report.	object
page	The current page number displayed with the report. This property is updated as the pages are viewed.	value: numeric
pageCount	A read-only property that provides the total number of pages in the report.	value: numeric
zoomLevel	A number representing the desired zoom level as a percentage of the report width. The Fit Panel option is simply 1 as opposed to a percentage value.	value: numeric
allowDownload	If set to 'true,' an icon will be added to the control bar at the bottom of the component that will download the report as a PDF file when clicked. The value of this property is always treated as "false" when the report viewer component is viewed from the iOS and Android Perspective Mobile apps.	value: boolean
downloadFilename	A filename used for the report if downloaded. Filename will default to the name of the report if this property is left blank.	value: string
allowOpenInTab	If set to 'true,' an icon will be added to the control bar at the bottom of the component that will open a temporary PDF file in a new tab. The value of this property is always treated as "false" when the report viewer component is viewed from Perspective Workstation, as well as the iOS and Android Perspective Mobile apps.	value: boolean
controlStyle	An object containing CSS style properties that are applied to the control bar and controls across the bottom.	object

Name	Description	Property Type
	• **classes**: Styles defined in the project to be applied to this component.	
style	An object containing CSS style properties that are applied to the background display of the component.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

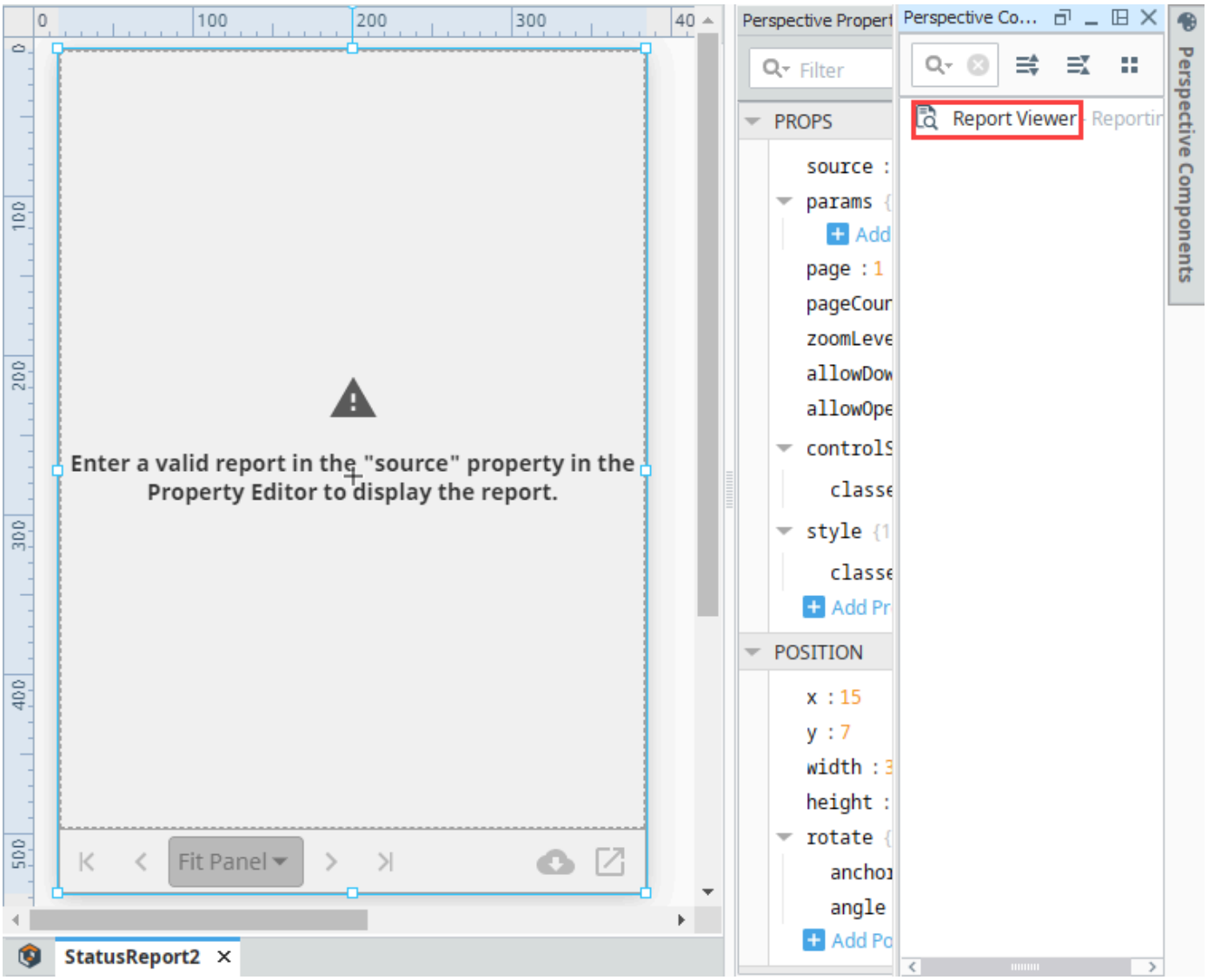
Example

Configuring the Report Viewer

The Report Viewer component provides an easy way to view and print reports in a Perspective View. Let's configure the Report Viewer for a report.

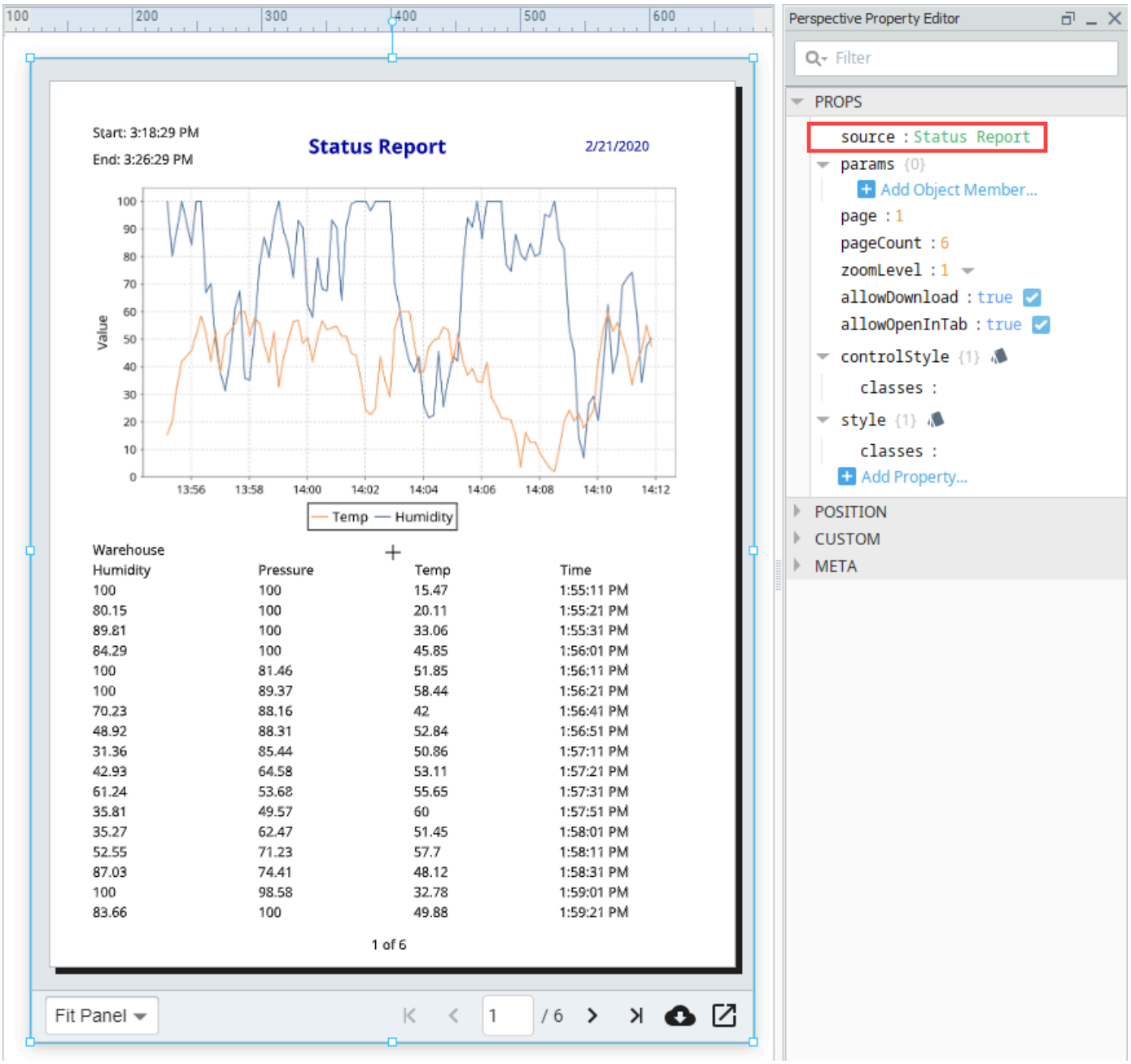
> **NOTE**
>
> This example requires that you already have a report created in your Gateway. You can learn more about creating reports here.

1. Drag a **Report Viewer** component into a view.

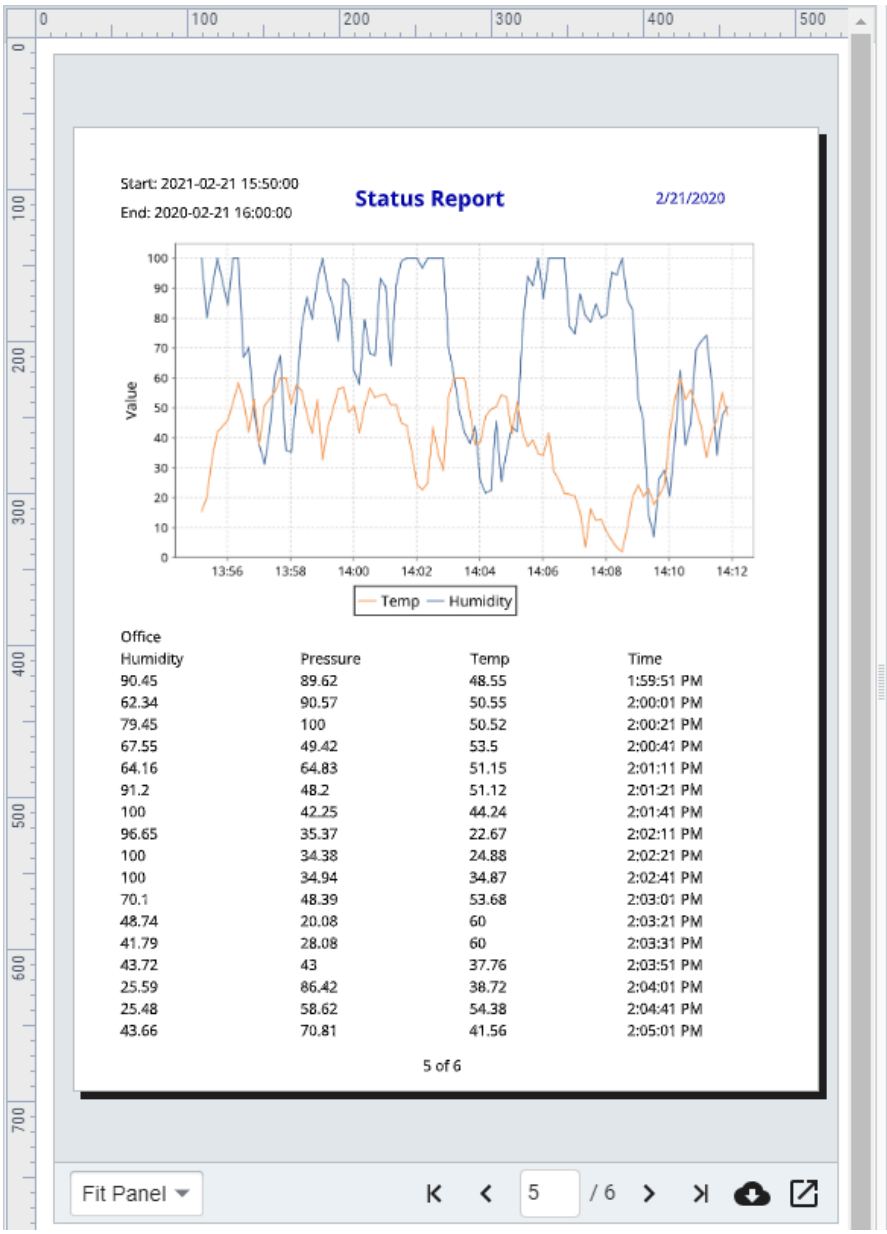


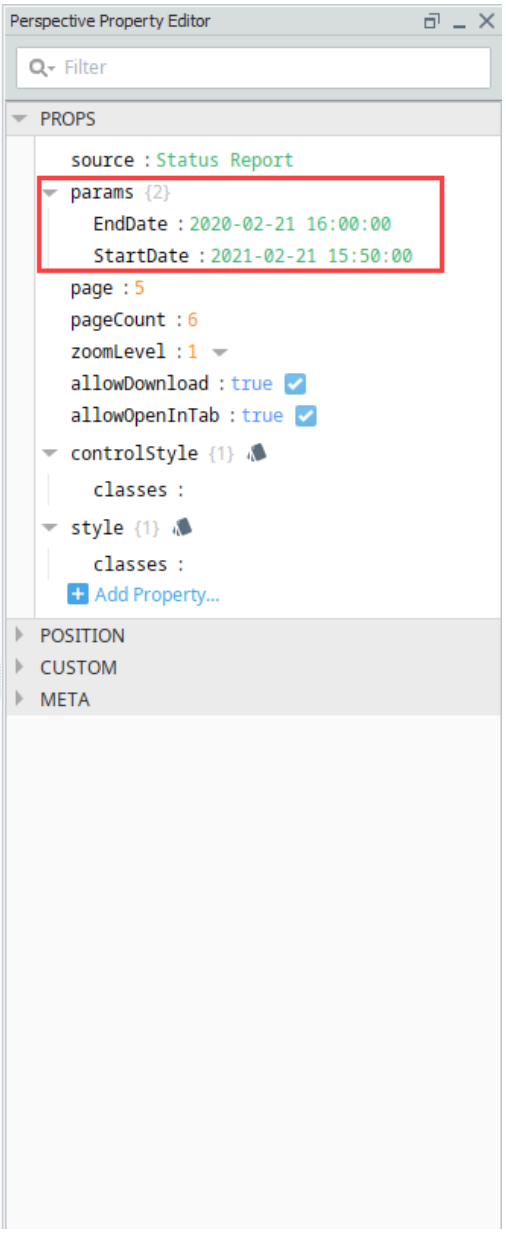
2. Enter the Path of the report you want to view in the '**source**' property of the Property Editor. The data from your report will immediately load into the Report Viewer.

> **INFO**
>
> You can right-click on any report in the designer Project Browser to Copy the full Path of the report.



3. Parameters added during report creation are provided as properties in the Report Viewer. Add them under the '**params**' object. (The example below uses the EndDate and StartDate parameters). The parameter names must match exactly to the parameters in your Report Resource, and will override any default values set in the Report Resource.


Start: 2021-02-21 15:50:00
End: 2020-02-21 16:00:00
Status Report
2/21/2020
Value
Temp
Humidity
Office
Humidity Pressure Temp Time
90.45 89.62 48.55 1:59:51 PM
62.34 90.57 50.55 2:00:01 PM
79.45 100 50.52 2:00:21 PM
67.55 49.42 53.5 2:00:41 PM
64.16 64.83 51.15 2:01:11 PM
91.2 48.2 51.12 2:01:21 PM
100 42.25 44.24 2:01:41 PM
96.65 35.37 22.67 2:02:11 PM
100 34.38 24.88 2:02:21 PM
100 34.94 34.87 2:02:41 PM
70.1 48.39 53.68 2:03:01 PM
48.74 20.08 60 2:03:21 PM
41.79 28.08 60 2:03:31 PM
43.72 43 37.76 2:03:51 PM
25.59 86.42 38.72 2:04:01 PM
25.48 58.62 54.38 2:04:41 PM
43.66 70.81 41.56 2:05:01 PM
5 of 6
Fit Panel
5
/ 6


Perspective Property Editor
Filter
PROPS
source : Status Report
params {2}
EndDate : 2020-02-21 16:00:00
StartDate : 2021-02-21 15:50:00
page : 5
pageCount : 6
zoomLevel : 1
allowDownload : true
allowOpenInTab : true
controlStyle {1}
classes :
style {1}
classes :
Add Property...
POSITION
CUSTOM
META

Perspective - Symbols Palette

The following symbol components provide you with design options for visualizing HMI. Click on the Symbol component name for a link pointing to a page containing the component's description, properties, and an example of how to configure it.

Motor

An animated component that looks like a motor.

Pump

An animated component that looks like a pump.

Sensor

An animated component that looks like a sensor.

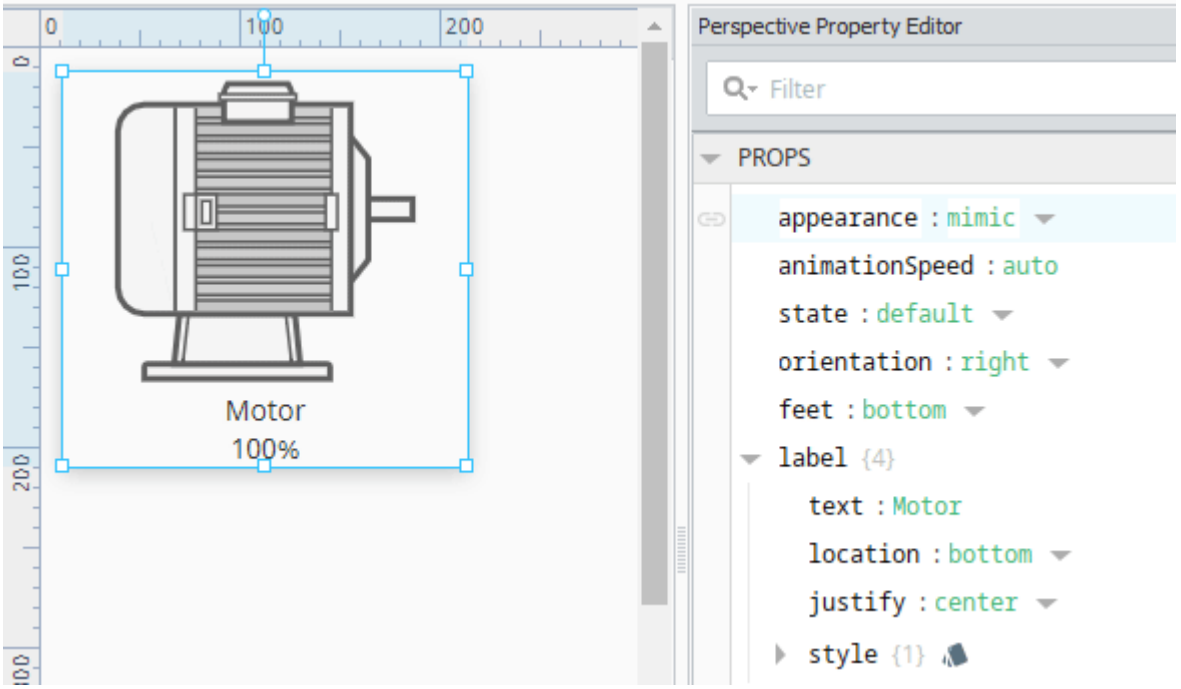
Valve

A component that looks like a valve.

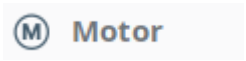
Vessel

An animated component that looks like a tank or vessel. Options include displaying fill level, the stand, and the a…

Perspective - Motor



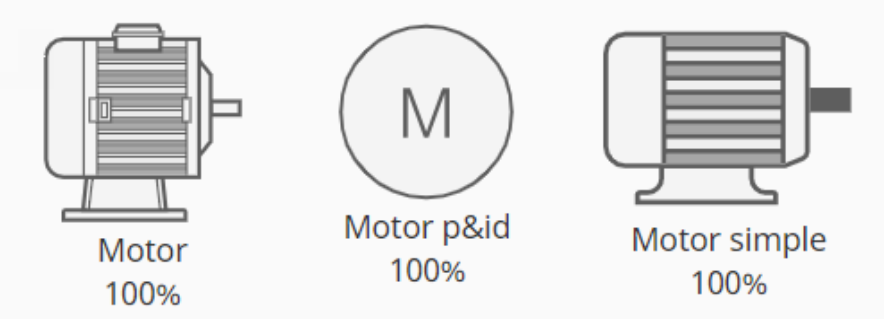
Component Palette Icon:



An animated component that looks like a motor. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

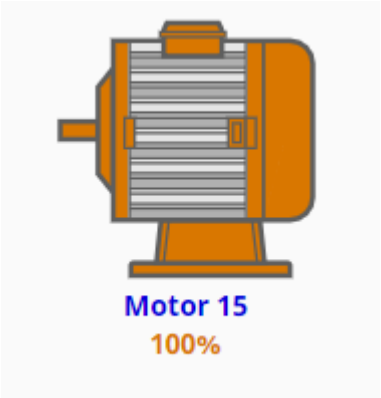
Name	Description	Property Type
appearance	Options for the appearance of the component. Options are auto, p&id, mimic, and simple. If set to auto, the value is taken from the Perspective **Session Property** symbols.autoAppearance. 	value: string
animationSpeed	The speed of the animation as a percent. Set to "0" to turn off animation. If set to auto, the setting is taken from the Perspective **Session Property** symbols.autoAnimationSpeed.	value: numeric
state	State of the animation. Built-in options are default, running, stopped, or faulted. Default is default. Built-in states and new states can be configured and applied on the **Project Properties > Symbols** page.	value: string
orientation	Orientation of the component. Options are top, bottom, left, or right. Default is right.	color
label	Label settings for the component. • **text**: Text for the label. Value is string. • **location**: Label position relative to the motor: top, bottom, left, right, or hidden. Default is bottom. Value is string. • **justify**: Label text justification: left, center, right, or auto. Default is center. Value is string. • **style**: Modify label text style using the style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**	object

Name	Description	Property Type
value	Value settings for the component. • **text**: Value to display as text. Value is string. • **location**: Value position relative to the motor: top, bottom, left, right, or hidden. Default is bottom. Value is string. • **justify**: Value text justification: left, center, right, or auto. Default is center. Value is string. • **style**: Sets a style for the value Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**	object
style	Sets a style for this cylindrical tank. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. The Pump has two pre-configured **variants**.	object

Component Events

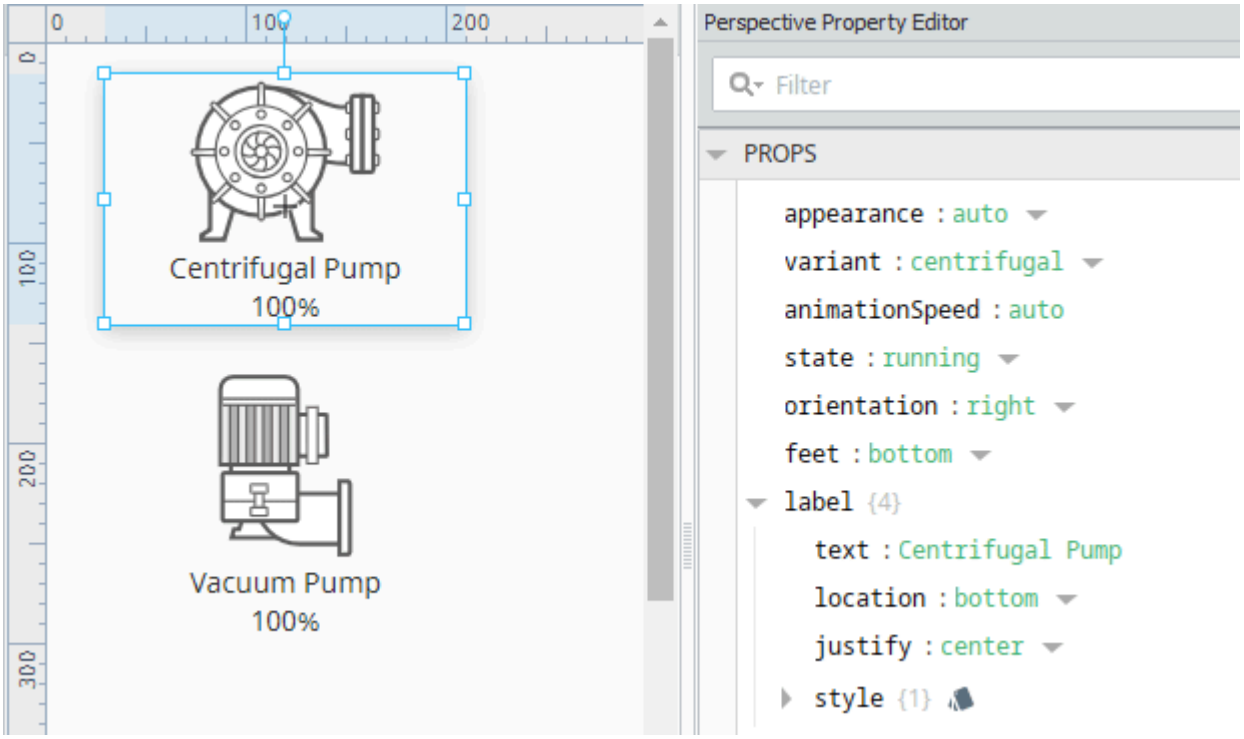
The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

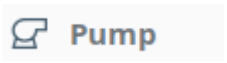


Property	Value
props.orientation	left
props.label.text	Motor 15
props.label.style.color	#0000D9
props.label.style.fontWeight	bold
props.label.style.fontFamily	Noto Sans
props.value.style.color	#D97700
props.value.style.fontWeight	bold
props.value.style.fontFamily	Noto Sans
props.style.fill	#D97700

Perspective - Pump



Component Palette Icon:

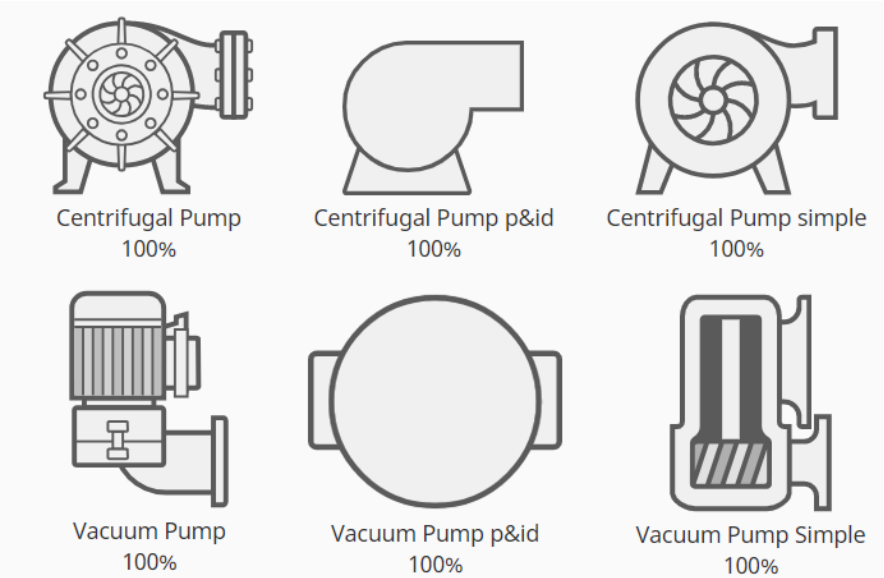


An animated component that looks like a pump. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. The Pump has two pre-configured **variants**:

- **Centrifugal**: Component appearance is that of a centrifugal pump.
- **Vacuum**: Component appearance is that of a vacuum pump.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

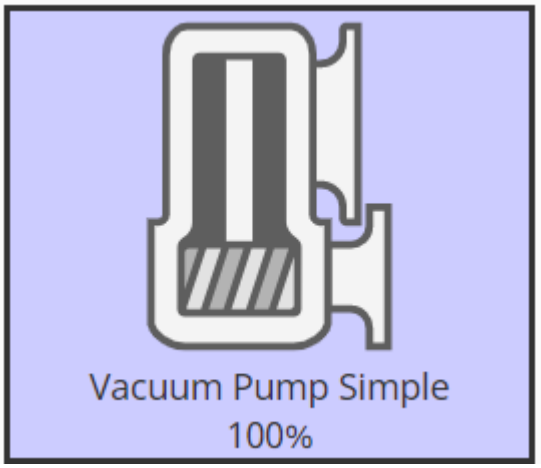
Name	Description	Property Type
appearance	Options for the appearance of the component. Options are auto, p&id, mimic, and simple. If set to auto, the value is taken from the Perspective **Session Property** symbols.autoAppearance. 	value: string
variant	Variant of pump to display. Options are centrifugal or vacuum. Default is centrifugal.	value: string
animationSpeed	The speed of the animation as a percent. Set to "0" to turn off animation. If set to auto, then the animationSpeed setting is taken from the Perspective **Session Property**.	value: numeric
state	State of the animation. Built-in options are default, running, stopped, or faulted. Default is default. Built-in states and new states can be configured and applied on the **Project Properties > Symbols** page.	value: string
orientation	Orientation of the pump. Options are top, bottom, left, or right. Default is right.	color
feet	Feet location for the pump. Options are top, bottom, left, or right. Default is bottom.	
label	Label settings for the component. • **text**: Text for the label. Value is string.	object

Name	Description	Property Type
	• **location**: Label position relative to the pump: top, bottom, left, right, or hidden. Default is bottom. Value is string. • **justify**: Label text justification: left, center, right, or auto. Default is center. Value is string. • **style**: Modify label text style using the style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**	
value	Value settings for the component. • **text**: Value to display as text. Value is string. • **location**: Value position relative to the pump: top, bottom, left, right, or hidden. Default is bottom. Value is string. • **justify**: Value text justification: left, center, right, or auto. Default is center. Value is string. • **style**: Sets a style for the value Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**	object
style	Sets a style for this cylindrical tank. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**. The Pump has two pre-configured **variants**.	object

Component Events

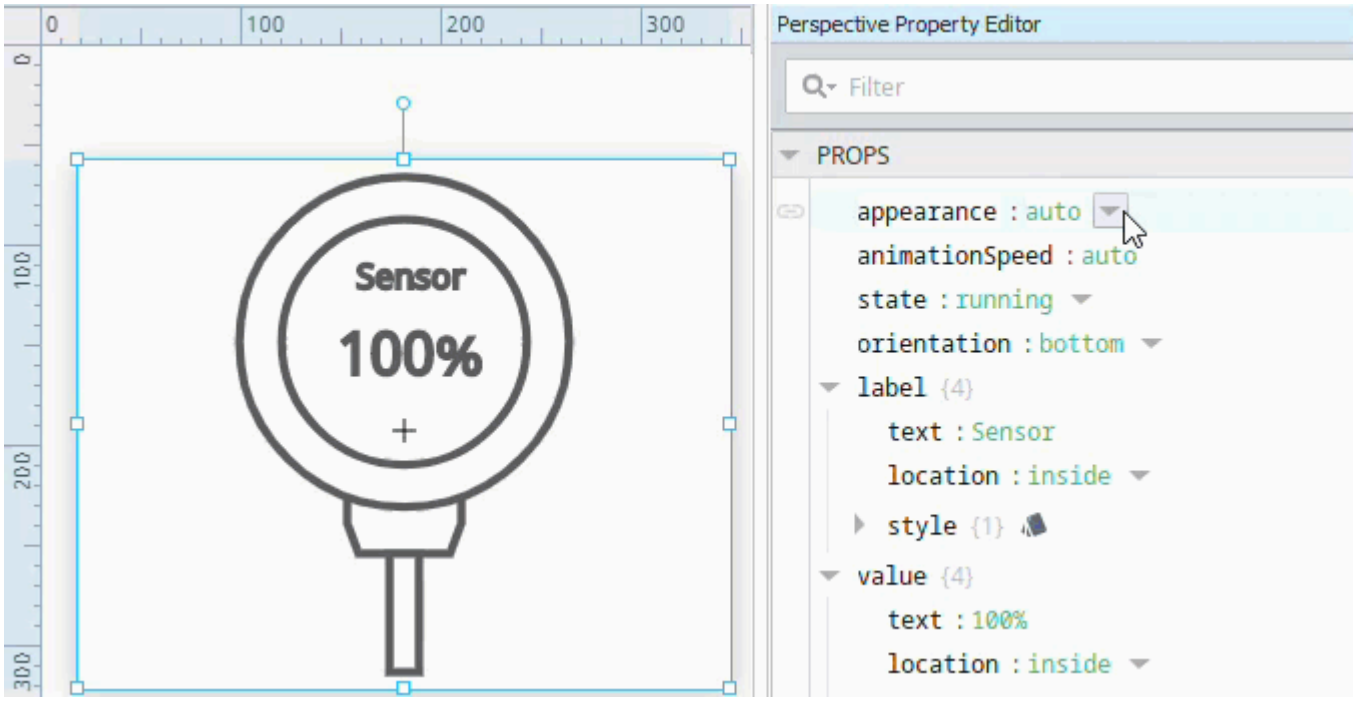
The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Example

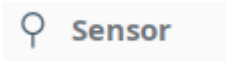


Property	Value
props.appearance	simple
props.variant	vacuum
props.label.text	Vacuum Pump Simple
props.style.backgroundColor	#CCCCFF
props.style.borderStyle	solid

Perspective - Sensor



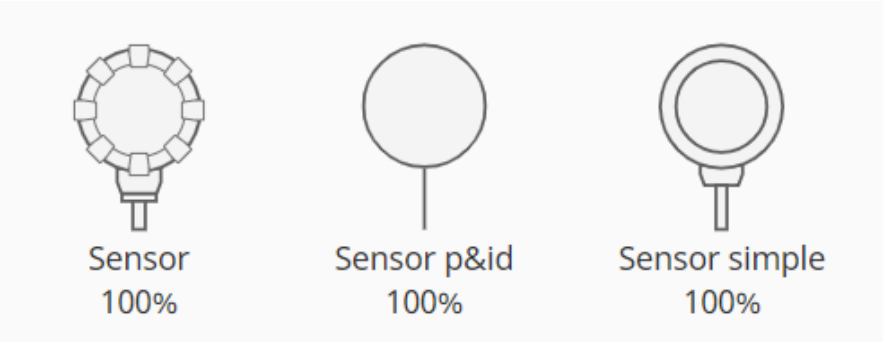
Component Palette Icon:



An animated component that looks like a sensor. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
appearance	Options for the appearance of the component. Options are auto, p&id, mimic, and simple. If set to auto, the value is taken from the Perspective **Session Property** symbols.autoAppearance. 	value: string
animationSpeed	The speed of the animation as a percent. Set to "0" to turn off animation. If set to auto, the setting is taken from the Perspective **Session Property** symbols.autoAnimationSpeed.	value: numeric
state	State of the animation. Built-in options are default, running, stopped, or faulted. Built-in states and new states can be configured and applied on the **Project Properties > Symbols** page.	value: string
orientation	Orientation of the sensor. Options are top, bottom, left, or right. Default is right.	value: string

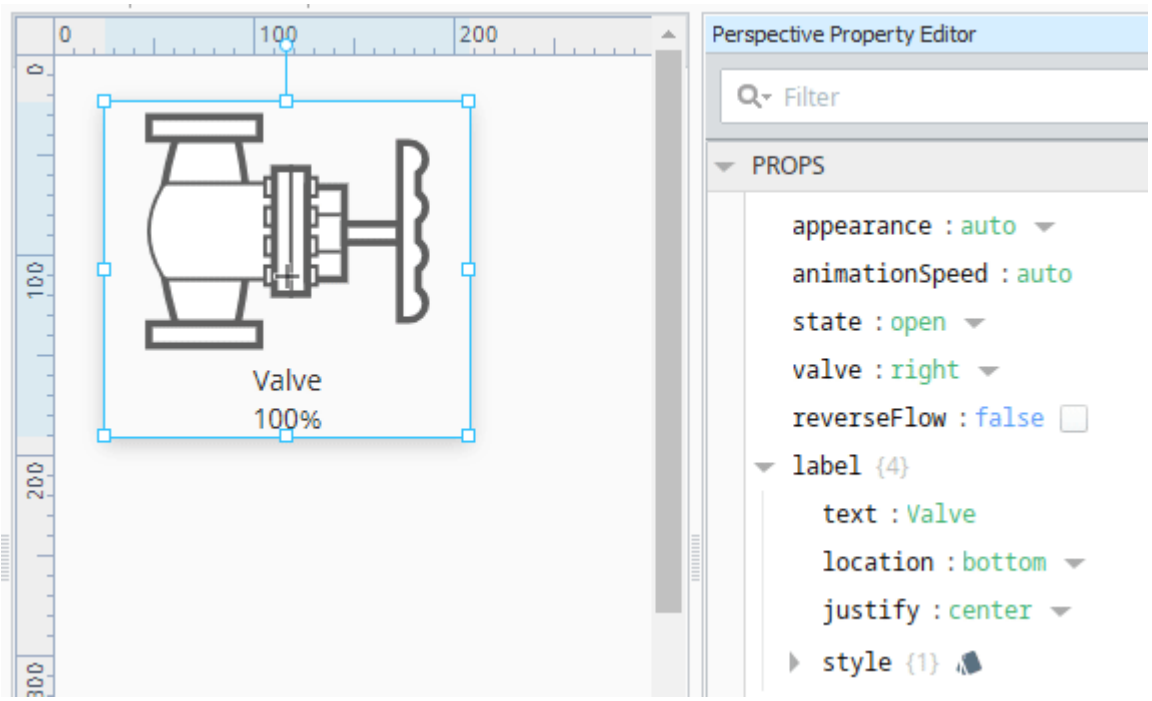
Name	Description	Property Type
label	Label settings for the component. • **text**: Text for the label. Value is string. • **location**: Label position relative to the sensor: top, bottom, left, right, or hidden. Default is bottom. Value is string. • **justify**: Label text justification: left, center, right, or auto. Default is center. Value is string. • **style**: Modify label text style using the style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**	object
value	Value settings for the component. • **text**: Value to display as text. Value is string. • **location**: Value position relative to the sensor: top, bottom, left, right, or hidden. Default is bottom. Value is string. • **justify**: Value text justification: left, center, right, or auto. Default is center. Value is string. • **style**: Sets a style for the value Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**	object
style	Sets a style for this cylindrical tank. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The

Component Events and Actions page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Perspective - Valve



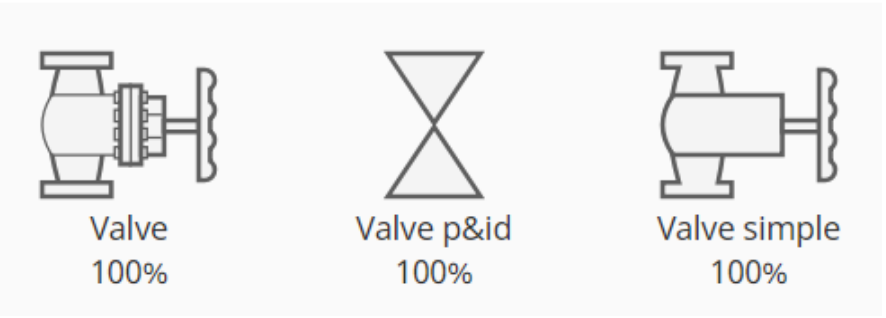
Component Palette Icon:



A component that looks like a valve. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

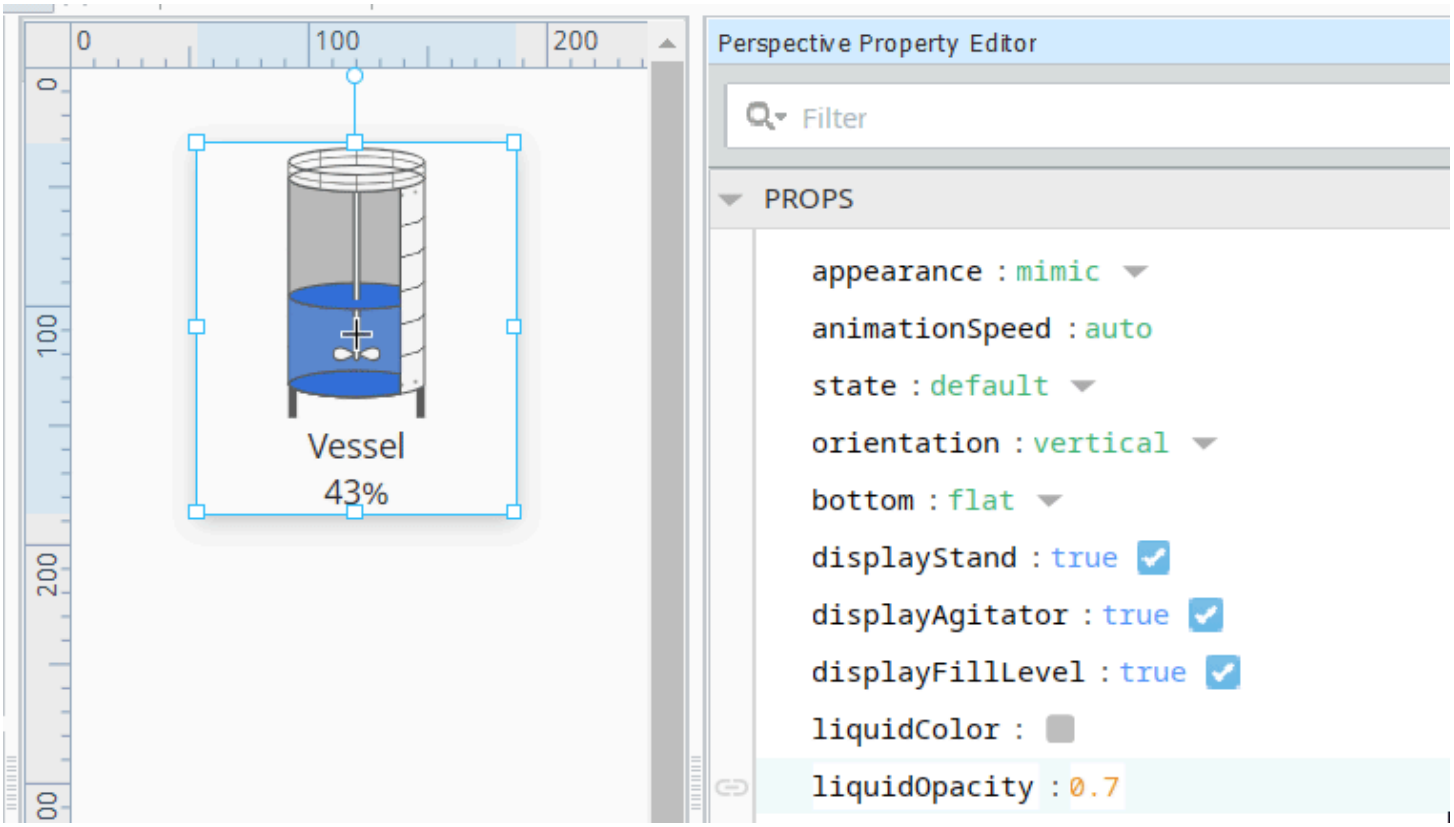
Name	Description	Property Type
appearance	Options for the appearance of the component. Options are auto, p&id, mimic, and simple. If set to auto, the value is taken from the Perspective **Session Property** symbols.autoAppearance. 	value: string
animationSpeed	The speed of the animation as a percent. Set to "0" to turn off animation. If set to auto, the setting is taken from the Perspective **Session Property** symbols.autoAnimationSpeed.	value: numeric
state	State of the valve. Built-in options are default, open, failedToOpen, partiallyClosed, closed, or failedToClose. Built-in states and new states can be configured and applied on the **Project Properties > Symbols** page.	value: string
valve	Orientation of the valve. Options are top, bottom, left, or right. Default is right.	value: string
reverseFlow	Whether or not to reverse the direction of the flow. Default is false.	value: boolean
label	Label settings for the component. • **text**: Text for the label. Value is string. • **location**: Label position relative to the valve: top, bottom, left, right, or hidden. Default is bottom. Value is string. • **justify**: Label text justification: left, center, right, or auto. Default is center. Value is string. • **style**: Modify label text style using the style properties. Full menu of **style options** is available for text,	value: numeric

Name	Description	Property Type
	background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**	
value	Label settings for the component. • **text**: Value to display as text. Value is string. • **location**: Value position relative to the valve: top, bottom, left, right, or hidden. Default is bottom. Value is string. • **justify**: Value text justification: left, center, right, or auto. Default is center. Value is string. • **style**: Sets a style for the value Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**	color
style	Sets a style for this cylindrical tank. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

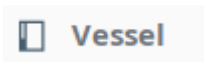
Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

Perspective - Vessel



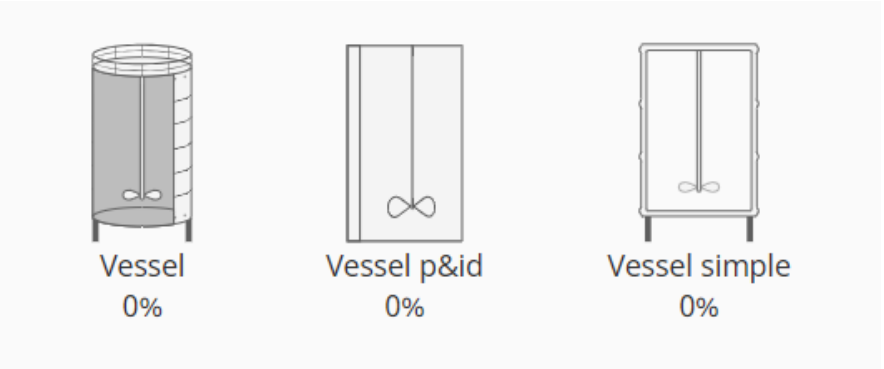
Component Palette Icon:



An animated component that looks like a tank or vessel. Options include displaying fill level, the stand, and the agitator. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.

Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
appearance	Options for the appearance of the component. Options are auto, p&id, mimic, and simple. If set to auto, the value is taken from the Perspective **Session Property** symbols.autoAppearance. 	value: string
animationSpeed	The speed of the animation as a percent. Set to "0" to turn off animation. If set to auto, the setting is taken from the Perspective **Session Property** symbols.autoAnimationSpeed.	value: numeric
state	State of the animation. Built-in options are default, running, stopped, or faulted. Default is default. Built-in states and new states can be configured and applied on the **Project Properties > Symbols** page.	value: string
orientation	Orientation of the vessel. Options are horizontal or vertical. Default is vertical.	value: string
displayStand	Whether or not to display the stand for the vessel. Default is true.	value: boolean
displayAgitator	Whether or not to display the agitator in the vessel. Default is true.	value: boolean
displayFillLevel	Whether or not to display the fill level of liquid in the vessel. Default is true.	value: boolean

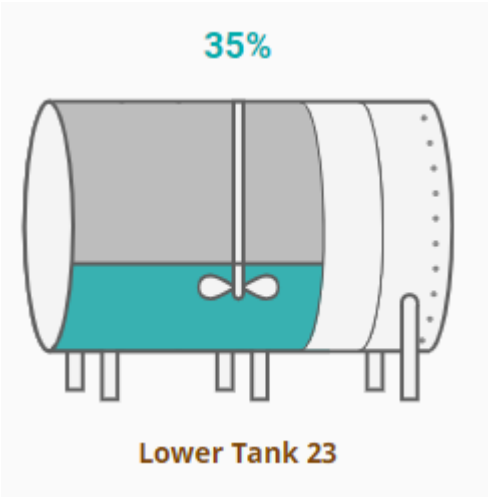
Name	Description	Property Type
liquidColor	Color used to render the filled part of the vessel. Can be chosen from color wheel, chosen from color palette, or entered as RGB or HSL value. See **Color Selector**.	color
liquidOpacity	The opacity of the liquid in the tank. 0 is fully transparent, 1 is fully opaque. Default is 0.7.	value: numeric
liquidWarningColor	The warning color of the liquid in the tank. See **Color Selector**.	color
label	Label settings for the component. • **text**: Text for the label. Value is string. • **location**: Label position relative to the vessel: top, bottom, left, right, or hidden. Default is bottom. Value is string. • **justify**: Label text justification: left, center, right, or auto. Default is center. Value is string. • **style**: Modify label text style using the style properties. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**	object
value	Value settings for the component. • **displayValueAsPercent** Whether or not to display the value as a percentage. Value is boolean. • **value** Text to display as value. Value is string. • **capacity** Total capacity of the vessel. Default is 100. Value is numeric. • **location** Value location relative to the vessel: top, bottom, right, or left. Default is bottom. Value is string. • **justify** Value text justification: center, left, or right. Default is center. Value is string. • **style**: Sets a style for the value. Full menu of **style options** is available for text, background, margin and	object

Name	Description	Property Type
	padding, border, shape and miscellaneous. You can also specify a **style class**.	
style	Sets a style for this cylindrical tank. Full menu of **style options** is available for text, background, margin and padding, border, shape and miscellaneous. You can also specify a **style class**.	object

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.

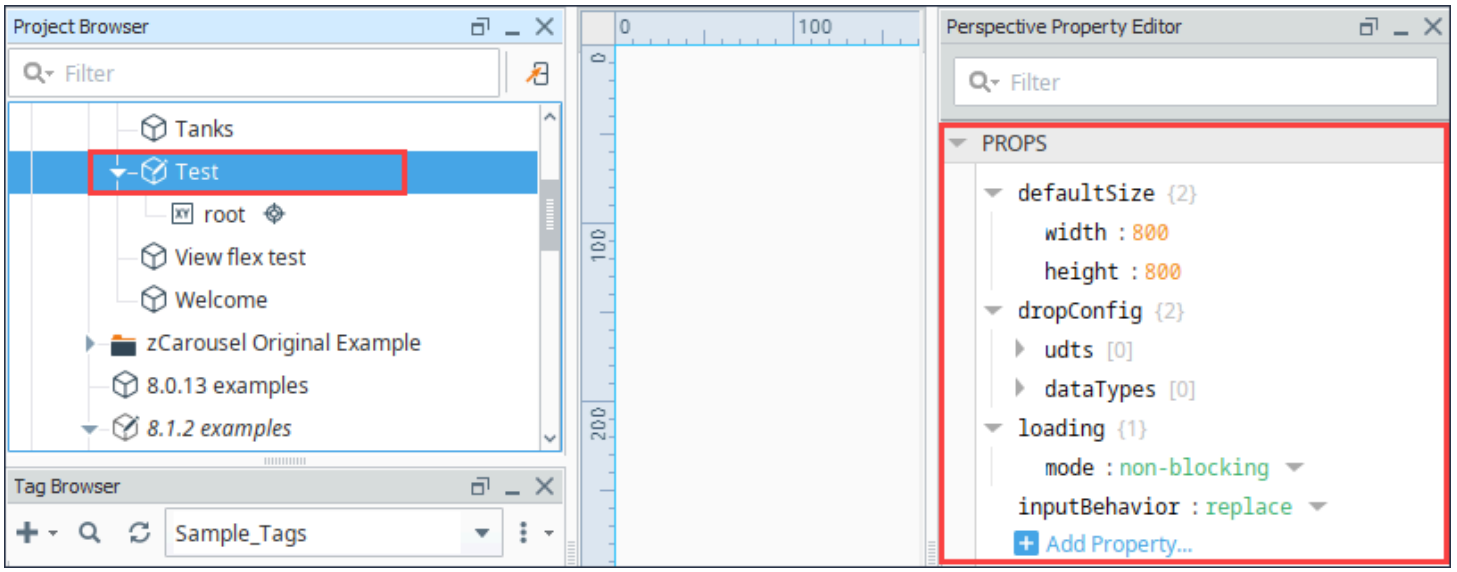
Example



Property	Value
props.orientation	horizontal
props.displayStand	true
props.liquidColor	#00ACAC
props.label.text	Lower Tank 23
props.label.location	bottom
props.label.style.fontWeight	bold
props.label.style.color	#804600
props.value.value	35
props.value.capacity	100
props.value.location	top
props.value.style.color	#00ACAC
props.value.style.fontFamily	Roboto
props.value.style.fontSize	18
props.value.style.fontWeight	bold

Perspective - View Object

This is the view itself. A view has properties that control the view size, defines how tags are dropped into a view, and settings for loading client views. The view properties are the same for each container type. View properties are displayed in the Perspective Property Editor when you click on the view in the Project Browser.



Properties

Most Properties have binding options. For more information on Bindings, see **Types of Bindings in Perspective**. This section only documents the Props Category of properties. The other Categories are described on the **Perspective Component Properties** page.

Name	Description	Property Type
defaultSize	Default size of the container. • **width**: Width of the view. Default is 800px. Value is numeric. • **height**: Height of the view. Default is 800px. Value is numeric.	value: numeric
dropConfig	Provides an opportunity to automatically create an instance of this view via tag drop. Click to see **dropConfig** properties. See the **Drop Configuration** page for more information.	object
loading	Options for loading the view. • **mode**: Provides two options for loading views, particularly for views with a high number of components: blocking and non-blocking. Blocking loads faster for views with fewer components. Non-blocking loads views with a large number of components in chunks. Default is 'non-blocking'. Views set to non-blocking will only wait patiently for five seconds if they have not been called. Value is string.	object
inputBehavior	Controls whether the internal properties of object-typed input and bi-directional parameters are merged with existing default values or whether the object-typed parameter should be strictly replaced with the supplied values. If "replace" is selected, and an object-typed parameter is supplied which does not contain a key for a default value, then that default value will not be used. Default is replace. **Examples** are included below.	object

dropConfig Properties

Name	Description	Property Type
udts	These settings allow for the automatic creation of this view when a UDT instance is dragged onto another view, using _this_ view to represent the UDT. A parameter is passed into the view, allowing the view to reference the UDT. • **type**: The name of the UDT to associate with this view. Value is string. • **param**: Name of the parameter on this view that the UDT member values will be passed to. Value is string. • **action**: Value is boolean. ◦ **bind**: Automatically configures a tag binding on the parameter (specified by **param**) to the UDT. ◦ **path**: Passes in a tag path string on the parameter (specified by **param**)	object
dataTypes	These settings allow for the automatic creation of this view when a standard tag is dragged onto another view. A parameter is passed into the view, allowing the view to reference the tag. • **type**: The tag data type to associate with this view. Value is string. • **param**: Name of the parameter to pass into this view. Value is string. • **action**: Value is boolean. ◦ **bind**: Automatically configures a tag binding on the parameter (specified by **param**) to the tag. ◦ **path**: Passes in a tag path string on the parameter (specified by **param**)	object

inputBehavior Examples

Defined “default” parameter:

```
myObject: {
    myParamOne: "One",
    myParamTwo: "Two"
}
```

Supplied parameter from Embedding Component:

```
myObject: {
    myParamOne: "1",
    myParamThree: "3"
}
```

If “merge” is selected:

```
myObject: {
    myParamOne: "1",
    myParamTwo: "Two",
    myParamThree: "3"
}
```

If “replace” is selected:

```
myObject: {
    myParamOne: "1",
    myParamThree: "3"
}
```

Note that there is no value for myParamTwo, so any references to this property will fault.

Component Events

The **Perspective Event Types Reference** page describes all the possible component event types for Perspective components. Not all component events support each Perspective component. The **Component Events and Actions** page shows how to configure events and actions on a Perspective component. Component scripting is handled separately and can be accessed from the Component menubar or by right clicking on the component.