Baudrate Data Analytics and Visualisation (DAV) Library

User Guide

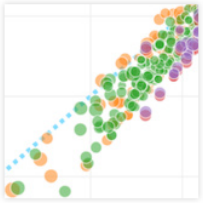
Contents

1 Introduction . . . 4
1.1 Key Features . . . 4
1.2 Requirements . . . 5
1.3 Installation . . . 5
1.4 Next Steps . . . 5

2 Component Guide . . . 6
2.1 Dav Service . . . 6
2.2 Widget Data Structure . . . 6
2.3 Attribute Editor . . . 7
2.4 Animation . . . 9
2.4.1 Format . . . 9
2.5 Ord Types . . . 11
2.5.1 Component Ord . . . 11
2.5.2 History Ord . . . 12
2.5.3 BQL and NEQL Ords . . . 13
2.5.4 Series Transform Ord . . . 13
2.6 Templates . . . 15
2.7 DAV Ord Scheme . . . 17
2.8 Relativization and Parameterization . . . 17
2.8.1 Relativization . . . 17
2.8.2 ORD Parameters . . . 18
2.8.3 Attribute Parameters . . . 19
2.9 Machine Learning . . . 19
2.9.1 Trendlines . . . 19
2.9.2 Forecasts . . . 19
2.9.3 DAV Copilot . . . 21
2.10 Additional resources . . . 24
2.10.1 Plotly.com . . . 24
2.10.2 Plotly Chart Studio . . . 24
2.10.3 Plotly.js JSON Editor . . . 24
2.10.4 Search Engines . . . 24
2.10.5 Single-Page Reference . . . 26

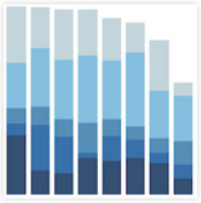
3 Step-by-step Guide . . . 28
3.1 Bar chart animated with numeric points . . . 28
3.2 Scatter chart animated with numeric history . . . 33

4 Attribute Reference 40
4.1 Trace Attributes 40
4.2 Layout Attributes 40
4.2.1 Frequently Used Layout Attributes 40
4.3 Config Attributes 40
4.3.1 Time Range Selector Buttons 41
4.3.2 Frequently Used Config Attributes 41

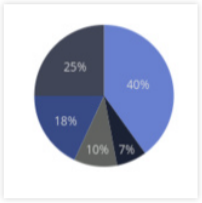


Scatter Plots

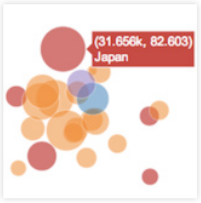
Line Charts



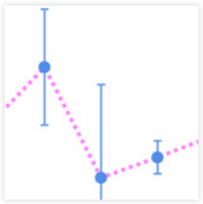
Bar Charts



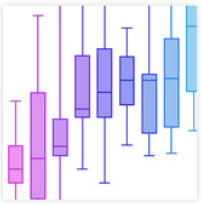
Pie Charts



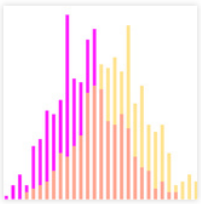
Bubble Charts



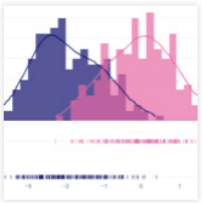
Error Bars



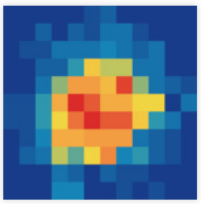
Box Plots



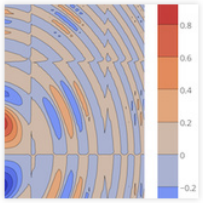
Histograms



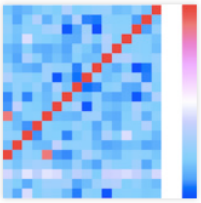
Distplots



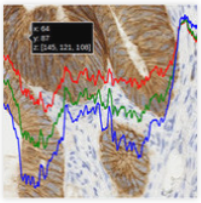
2D Histograms



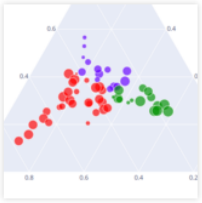
Contour Plots



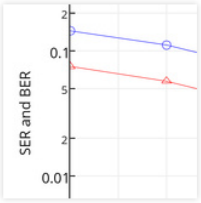
Heatmaps



Imshow



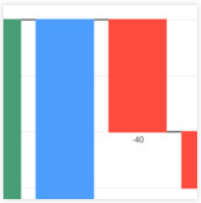
Ternary Plots



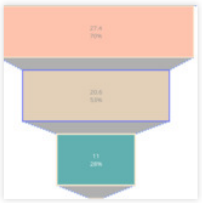
Log Plots

Time Series and Date Axes

Candlestick Charts

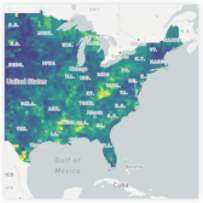


Waterfall Charts



Funnel Chart

OHLC Charts



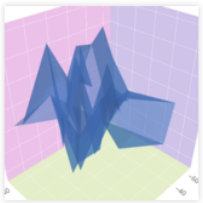
Mapbox Choropleth Maps

Lines on Mapbox

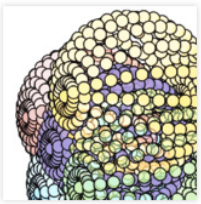
Filled Area on Maps

Bubble Maps

Mapbox Density Heatmap

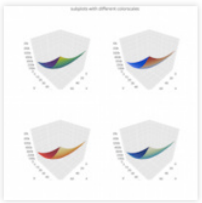


3D Axes

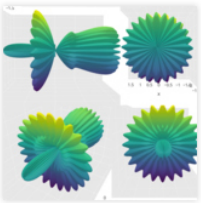


3D Scatter Plots

3D Surface Plots



3D Subplots



3D Camera Controls

1 Introduction

A picture is worth a thousand words, and certainly when it comes to data, metrics and KPIs, nothing could be more right. Without visual tools such as charts and graphs we would soon get lost in the numbers being presented to us. In building automation, visual presentation of data is a first-class citizen. With data collection and use continuing to increase exponentially, the need to visualize this data is becoming more important. Engineers seek to consolidate thousands of database records into beautiful charts and dashboards that humans can quickly and intuitively interpret.

Tridium Niagara offers a great platform to collect the data, and it comes with a few easy to use chart types that are sufficient to do a basic visualisation. However, when it comes to large amounts of data and more complex data presentation along with sophisticated dashboards – Baudrate Data Analytics and Visualisation (DAV) library can help.

The library is based on plotly.js, which is an interactive plotting library that supports over 40 unique chart types covering a wide range of statistical, financial, geographic, scientific, and 3-dimensional use-cases. Built on top of d3.js and stack.gl, plotly.js allows users to create beautiful interactive web-based visualisations that now can be displayed in standard Niagara PX files. Every chart type has hundreds of customisations options and almost every aspect of data visualisations can be modified to suit the needs. Plotly offers advanced charting capabilities which are used by leading data science and statistical companies. It allows engineers to put complex data analytics in the hands of business decision makers and operators.

Baudrate DAV library not just allow to insert plotly.js charts in Niagara PX views. It fully integrates all charts with Niagara framework with easy to use configuration UI, adds many ways to bind charts with real-time and historical data, relativize and parameterize charts, maintain consistent views with templates, set and switch time ranges for histories, introduce trend lines and forecasts, produce PDF reports, and much more.

1.1 Key Features

- over 40 unique chart types, including:
  - scatter: line, marker, area, bubbles
  - bars: grouped, stacked, overlay
  - pie: regular, pulled, donut
  - indicator: bullet, angular, datacard
  - heatmap: regular, contour
  - polar: bar, radar
  - sankey
  - sunburst
  - treemap
  - waterfall
  - geographical: scatter map, density map, choropleth
  - 3D: surface, scatter, mesh, volume
  - financial: OHLC, candlesticks, funnel
  - statistical: histogram, violin
- almost all chart aspects can be customized
- binding historical and real-time data
- BQL and NEQL queries
- absolute and relativized ords
- complex data queries with Series Transform
- interactive controls
- quick time range selection for historical data
- color schemes defined in templates

1.2 Requirements

- Tridium Niagara 4.8 or later powered device such as JACE, Supervisor or their OEM versions
- Baudrate DAV library license file

> **NOTE:** If you have previously used a different version of our library, it must be removed prior to installing the DAV library. Specifically, please delete the `plotly-ux.jar` file from the 'modules' directory before proceeding with the DAV library installation.

1.3 Installation

1. Copy **dav-ux.jar** and **dav-doc.jar** into /modules folder of Niagara Workbench, restart Workbench and Niagara station in Application Director.
2. If licensed for a JACE, install the modules via **Software Manager** and restart its station.
3. Open the palette and add **DavService** to **Services**.
4. Open the **DavService**. Press import icon » near **License** property and import the provided **.license** file. Press **Save** button and make sure **Status Message** shows "License ok".
5. Right-click on the service and select **Create Dummy Histories** action. This action creates a few histories to provide data for DAV library palette widgets.
6. Drag & drop any of the widget examples from the palette to the px file. You should see the widget with populated data.

1.4 Next Steps

The DAV library is shipped with the palette that contains examples of charts to get you started. Each palette example is bound to the dummy data source that can be re-bound to the actual data sources.

Note: this guide is available in two formats: as a PDF document and as a part of Niagara help file. Niagara help file includes interactive chart examples.

To understand main library concepts and components it is highly recommended to read the Component Guide section.

Then, in order to learn how to build charts from scratch, see Step-by-step Guide section.

Chats could be configured with lots of attributes. For a full description of all of them see Attribute Reference section.

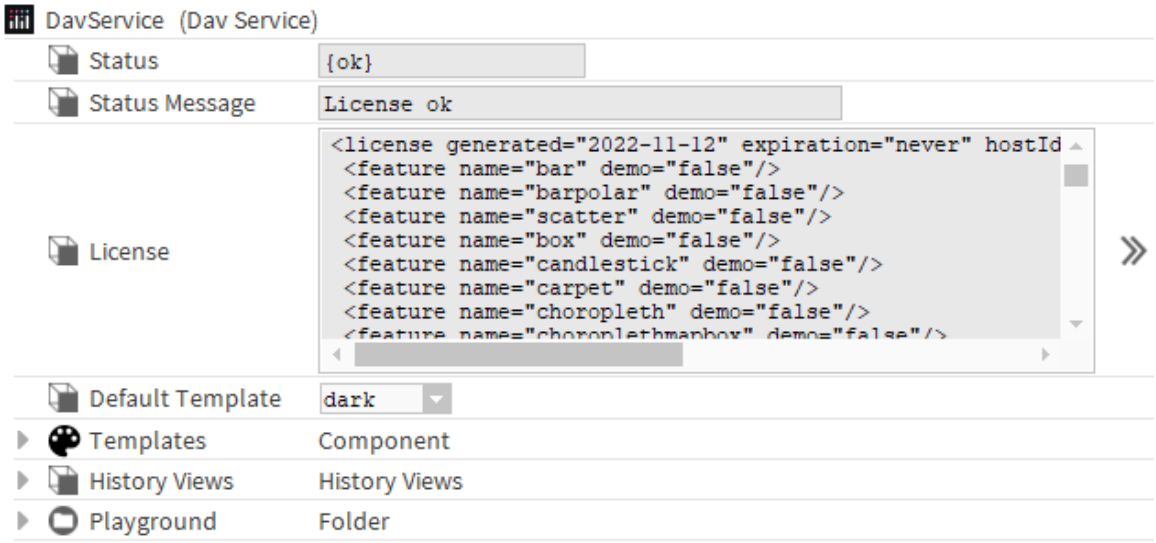


Figure 1: Dav Service

2 Component Guide

This section describes main library concepts and components one should know about in order to create new charts.

2.1 Dav Service

The station shall contain one **DavService** component installed in **Services** container. This component is responsible for licensing, templating and other features. It also stores test points used by DAV palette widgets. **DavService** properties:

- **Status** – `{ok}` if the service is enabled and licensed.
- **Status Message** – explains the status value, useful for debugging.
- **License** – displays license file content; press `>>` button to import the license file.
- **Default Template** – a combo-box to select the default template.
- **Templates** – a container with all templates, which can be applied to widgets. Please refer to Templates section.
- **Playground** – a folder with a few Niagara points and history generators to provide data for palette widgets.

DavService have **Create Dummy Histories** action to generate a few histories stored in **History/dav** container.

2.2 Widget Data Structure

DAV library charts are described declaratively as JSON objects. Every aspect of a DAV library chart (the colors, the grids, the data, and so on) has a corresponding JSON attribute. In Niagara each widget is configured with one or multiple **Dav Bindings** and two properties: **davLayout** (also referred as **layout**) and **config**.

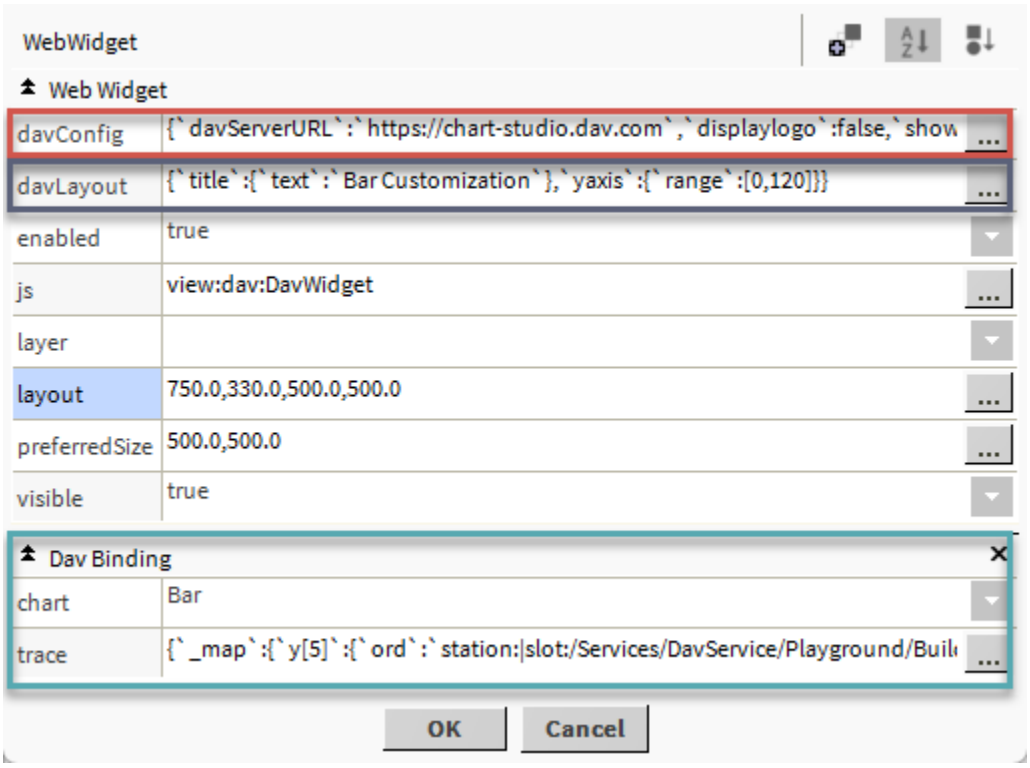


Figure 2: Niagara widget properties

- **Dav Bindings** are used to set widget type, data and type-specific attributes. Each binding has two properties: **chart** for type of the widget and **trace**, a set of attributes to configure various aspect of the chart. The attributes can be bound to real-time points or histories. They can be bound with various ord schemes: slot, history, bql, neql, seriesTransform etc. All attributes are described in Trace Attributes section.
- **davLayout** property defines attributes that apply to the rest of the chart, like title, xaxis, annotations. The detailed description is in Layout Attributes section.
- **davConfig** property defines high-level configuration options for the plot, such as the scroll / zoom / hover behaviour. The detailed description is in Config Attributes section. The difference between **config** and **davLayout** is that **davLayout** refers to the content of the plot, whereas **config** refers to the context in which the plot is being shown.
- **enabled**, **js**, **layer**, **layout**, **preferredSize**, **visible** are standard Niagara widget properties, please refer to Niagara documentation for more help.

2.3 Attribute Editor

Widget properties **trace**, **davLayout** and **davConfig** are collections of attributes describing the widget. Attributes are organized in a tree structure: one or multiple root attributes, which might have sub-attributes or children, which in turn might also have sub-attributes etc. As the total number of DAV library attributes is very high (in a range of hundreds), they are not shown initially. Instead, one has to add the attributes, which should be modified, and then set their values. It is not necessary to know each widget attribute. In fact, nobody knows that – there are just too many. It is also not needed, as all attributes by default have suitable values. So for a simple widget, which still will look

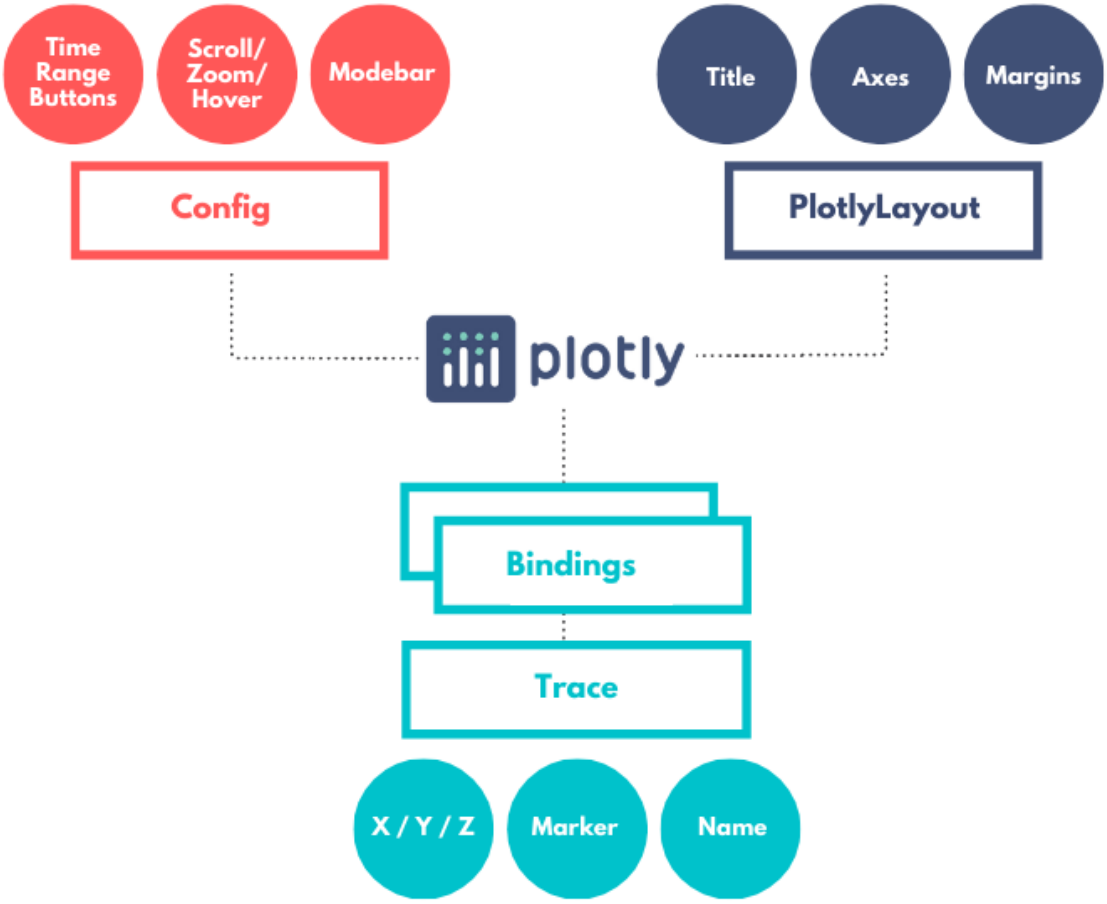


Figure 3: DAV object structure

good, only one or two attributes should be changed. By changing more attributes one can create very beautiful and interesting widgets.

When clicking on **trace**, **davLayout** or **davConfig** property, a new dialog box appears. Its top area contains added attributes, middle area contains buttons and bottom area has dynamic text, which describes selected attributes. This text helps to understand attribute purpose and possible values.

Note: **trace** attributes are different for every chart type, therefore **chart** property shall be defined first.

Buttons:

- Add Root [Ctrl+Insert] – to add trace root attribute – the ones on the top of a tree
- Add Child [Insert] – to add sub-attributes to selected attribute
- Remove [Delete] - to delete attribute
- Edit [Ctrl+E] - to modify attribute value
- Animate [Ctrl+A] - to animate attribute value using ord values
- Expand [Ctrl+Down] - expands all root properties so that sub-attributes are visible
- Collapse [Ctrl+Up] - collapses all root properties and their sub-attributes

Add Root opens a selector with all root attributes available for this object. The bottom area of the dialog box displays information of a selected attribute: its type, default value and description. After the attribute is added, it is possible to **Edit** its value. In case of complex attributes, it might be necessary to **Add Child** first. That will open the attribute selector containing all root children. Unused attributes can be deleted with **Remove** button.

2.4 Animation

Attributes of **trace** property can be animated – dynamically bound to Niagara points, components and histories. Animating property in Niagara means to link a widget property to a bound data component value, so that the widget can display any change in value as it occurs. In order to animate, use **Animate** button and enter **Ord** (using ord chooser or manually) and **Format**. Animated value will be displayed with a yellow background. To un-animate press on **Animate** button again.

Various types of ords and ord choosers are available:

- Component Chooser - to select ord to station component, which slot value should be displayed. For more information see Component Ord section.
- History Chooser – to select Niagara history, which time-series values should be displayed. In addition, the time range, rollup period and function can be specified. The history can also be reshaped into matrix. For more information see History Ord section.
- Transform Chooser – to select the Series Transform graph, which allows complex transformation of Niagara histories. For more information see Series Transform Ord section.
- BQL Query Builder – to run BQL query.
- NEQL Query – to run NEQL query (no query builder is available, should be entered manually). For more information see BQL and NEQL Ords section.

2.4.1 Format

In addition to the **Ord** property, the **Format** property is used to extract specific information from bound components and to do its processing. Format is based on a standard Niagara BFormat feature: a regular text with embedded scripts denoted by the % character. Calls within the script are mapped to an object's methods, and can be chained using the dot operator. It is also possible to embed mathematical operations, array operations and JavaScript function calls to modify values for widgets.

Examples for numeric points:

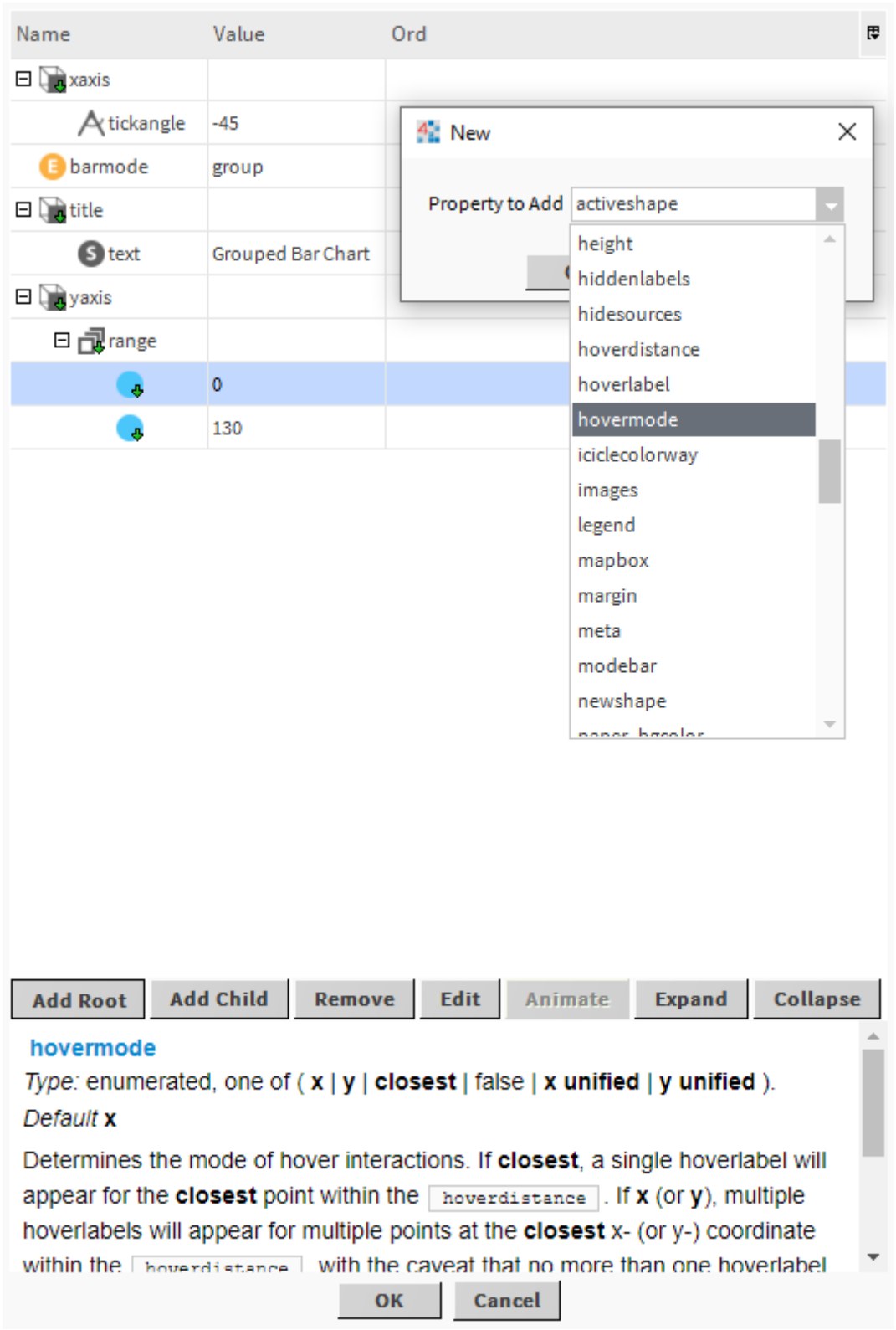


Figure 4: Attribute editor

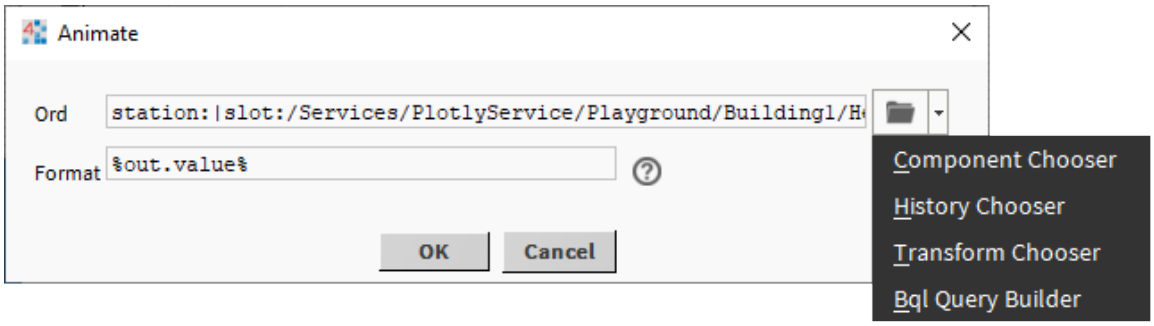


Figure 5: Animate dialog box

- `%out.value%` – returns value of the point to be used as data in widget attributes like `x`, `y`, `values`.
- `%displayName%` – returns user-friendly name of the point to be used in widget attributes like `labels`.
- `%anyName%` – returns value of extra property **anyName**, which can be added in the Niagara slot sheet.
- `%out.value% * 10` – returns out.value multiplied by 10.
- `%out.value% > 100 ? 'red' : 'blue'` – returns a red or blue text string depending on `out.value`. Can be used to dynamically change colors.

For Niagara tables (histories, series transform) **Format** property shall specify the name of table column:

- `%timestamp%` – returns history timestamps to animate attributes like `x` in Scatter widgets.
- `%value%` – returns values from non-rollup histories. Can be used for `y` attribute in Scatter widgets.
- `%max%`, `%min%`, `%avg%` – returns max, min and average values for each period in rollup history. Useful for `high` and `low` attributes in Candlestick widgets.
- `%first%`, `%last%` – returns first and last values for each period in rollup history. Useful for `open` and `close` attributes in OHLC widgets.

It is possible to apply JavaScript array functions to the whole history column:

- `%value%.slice(0,10)` – returns 10 first values from the value column.
- `%max%[0]` – returns the first value from the table max column as a scalar. Can be used in Indicator.
- `%min%[%min%.length-1]` – returns the last value from the min column.

There is a special syntax of **Format** allowing to process every value in the column separately: two percentage signs instead of one – `%%x%%`. Example:

- `(%%status%% & 8) == 8 ? 'red' : 'green'` – encode the alarm flag in the history status column as a color string to animate `marker.color`.

2.5 Ord Types

2.5.1 Component Ord

This is the basic ord, pointing to any Niagara station components, usually to numeric points. The widget subscribes to these components and displays their slots as specified by **Format**

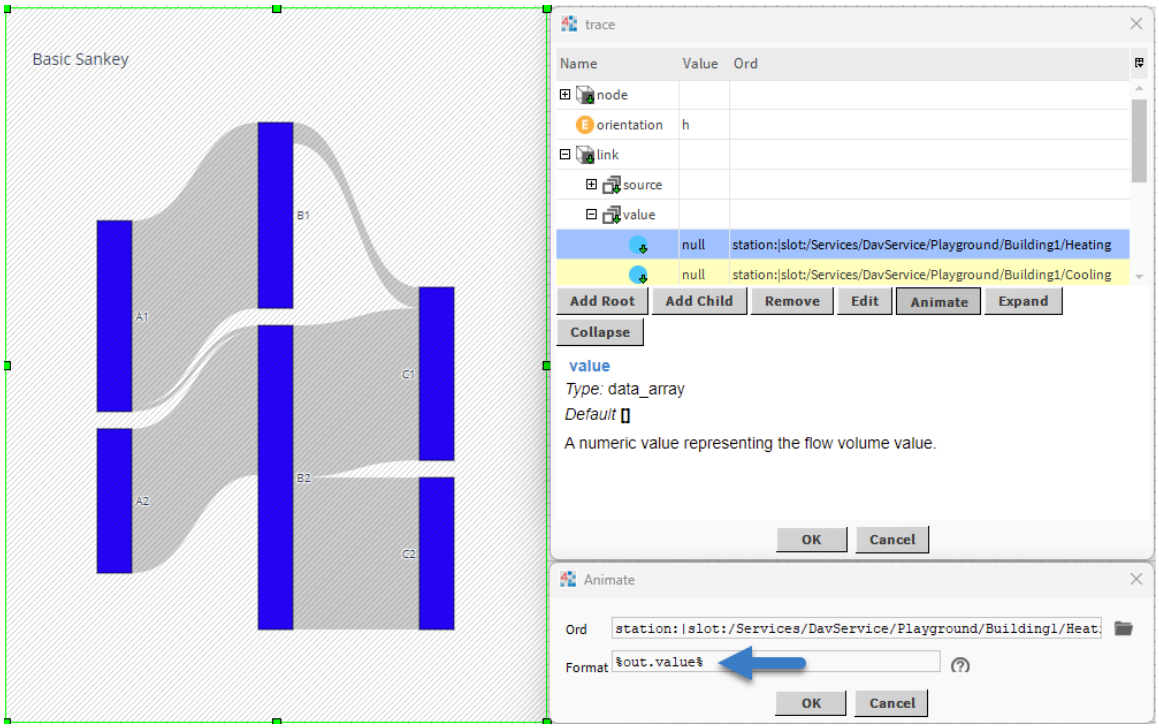


Figure 6: Slot Ord

property in real-time. For example ord `station:|slot:/Folder/NumericPoint1` subscribes to Folder/NumericPoint1.

2.5.2 History Ord

History ord points to Niagara history and also specifies a few extra parameters:

- **Time Range** - to specify default history record time range. There is a number of time ranges that can be selected from the dropdown menu:
  - Standard ones: **Today**, **Yesterday**, **This Week**, **Week To Date**, **Last Week**, **This Month**, **Month To Date**, **Last Month**, **This Year**, **Year To Date**, **Last Year**, **All**.
  - **Custom Range**: fixed time period.
  - **Current** and **Previous** periods: specify a certain number of minutes, hours, days, weeks, months and years including (Current) or excluding (Previous) the current one.
  - **From** selector: the most powerful one for very complex periods, like "go back 2 months and select 3 first weeks". This selector allows to easily specify multiple periods of the same history for comparison, e.g. overlay and compare 4 last weeks of the same history on the same line chart.
- **Rollup Interval** - specifies an interval of time used to determine what data is presented in your chart. Each displayed point represents a designated time interval before the specified plot time. A rollup value of 1 hour will present data at a granularity level of every one hour, while a rollup value of 15 minutes will show data for every 15 minutes of logged data. Rollups are very useful to reduce granularity – you can collect data very often (e.g. once every minute), but still display fast and clean charts for a long time period (e.g. a year). Without rollups there would be too many data points, which can render charts cluttered and slow.

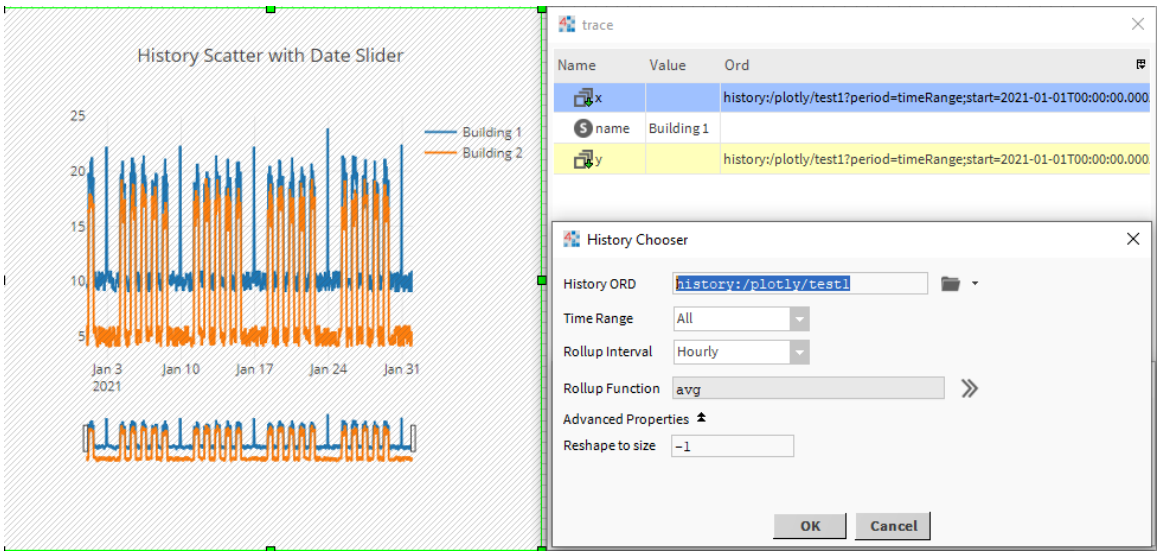


Figure 7: History Ord

- **Rollup Function** - specifies the functional operations against each record of the rollup period. Please note that Format used for animation will depend on the selected function. For example if you selected `avg` and `count` as your rollup functions you will be able to use them as *%avg%* and *%count%* to animate trace attributes. It is possible to use multiple rollup functions on the same chart – see examples in palette candlestick or OHLC charts.
- **Reshape to Size** - this property is used to transform one-dimensional history into a two-dimensional matrix, which is necessary for charts like heatmap or contour. It should be equal to the size of the first dimension of the matrix.

2.5.3 BQL and NEQL Ords

BQL and NEQL queries allow searching to Niagara components by name or by tags. The widget then subscribes to all found components and displays their values. This type of ords is great to make relativized universal views. For example, show a bar chart with all room temperatures on a certain floor. One widget will be able to display any number of temperature values, whether there are 5 or 50.

Both BQL and NEQL can be used for such a task. BQL queries rely on slot names and values: e.g. find all points whose names end with "Temp" and with alarm extension. Note: BQL queries should return component slotPath in the first column, to allow subscription.

NEQL queries rely on semantic tags – a more modern and flexible approach. For example query `station:|slot:/BuildingA/Level01|neql:n:point` will return all Niagara points stored in a folder /BuildingA/Level01.

2.5.4 Series Transform Ord

Series Transform is a standard Tridium Niagara module, which provides a way to manipulate Niagara histories, in order to create new series data. Series Transform is not well-known among the integrator community, although its graphs compensate for a missing JOIN operator in the Niagara BQL language. With Series Transform one can, for example, merge two or more histories using their timestamps. The resulting query produces a set of columns, each of which can be bound to a widgets attribute.

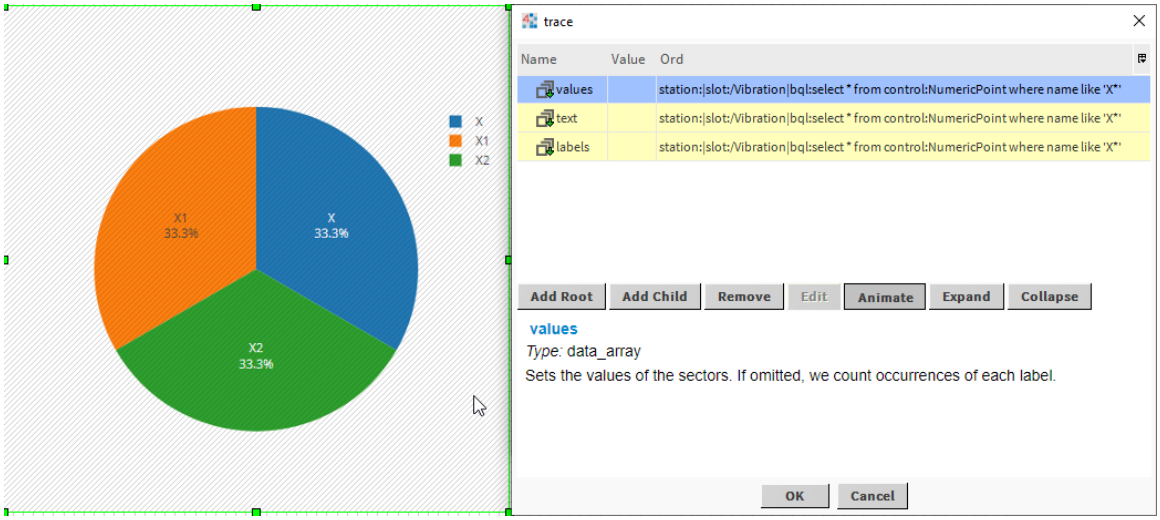


Figure 8: BQL Ord

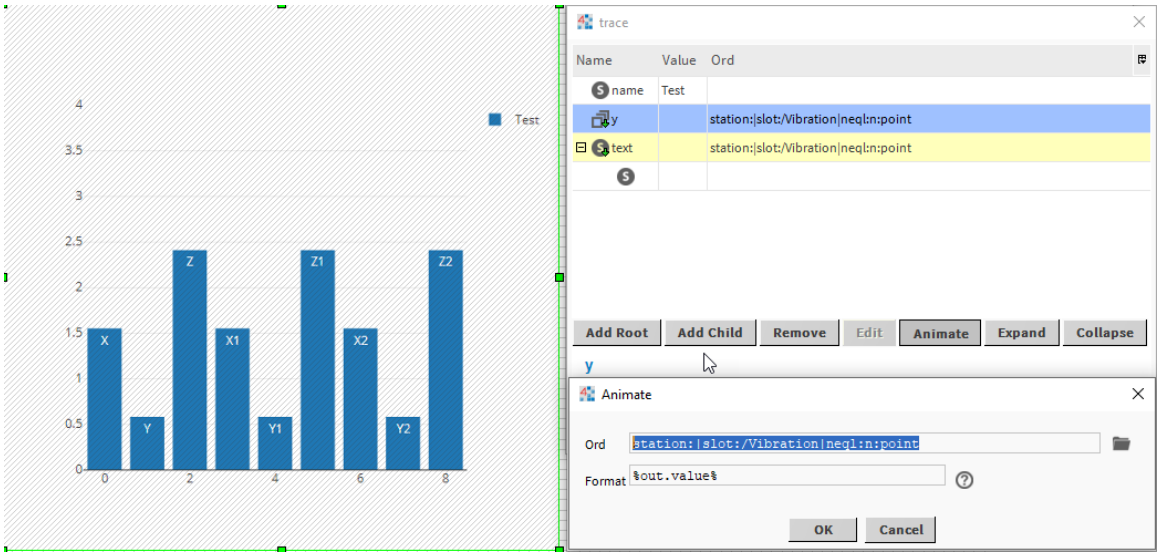


Figure 9: NEQL Ord

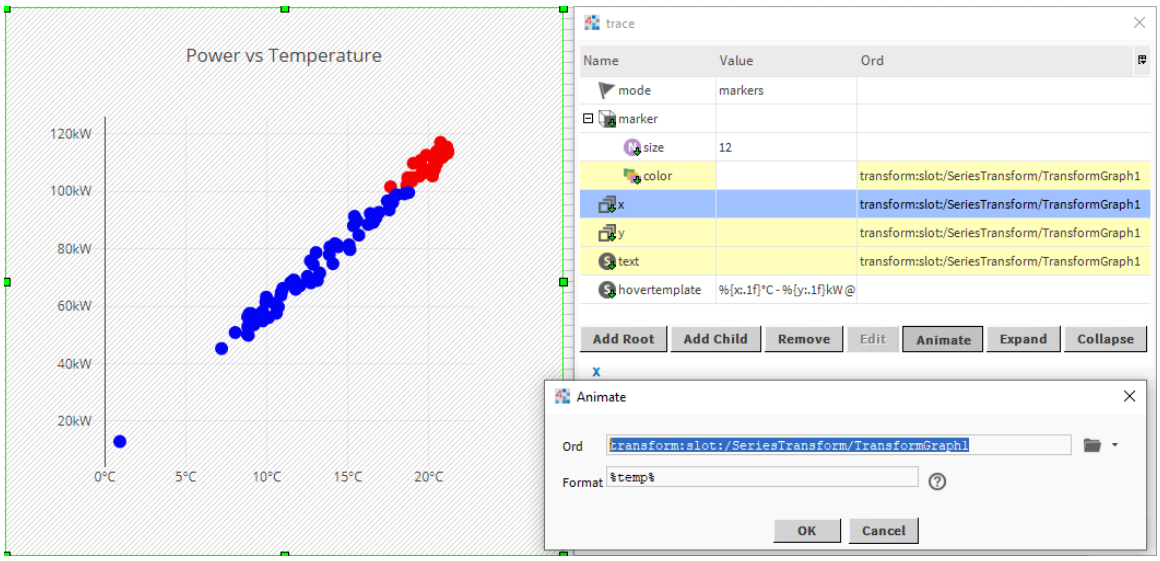


Figure 10: Series Transform Ord

Ord example: `transform:slot:/Folder/TransformGraph1` – assuming this graph combines outdoor temperature history and energy consumption history, its result can be used to draw XY scatter showing how energy consumption (Y) depends on outdoor temperature (X).

For more information about Series Transform, search for it in Tridium Niagara help.

2.6 Templates

Templates are powerful tools to maintain and update graphical design of charts, including their color scheme. The library is shipped with nine predefined templates and users can quickly customize them or make their own template to preserve company brand or to make fast changes to all widgets at once to suit customer wishes.

Each template is a collection of attributes to set default values for **trace**, **davLayout** and **davConfig** objects. The **Templates** property under **DavService** contains a set of named templates. Templates are editable with the standard attribute editor, which allows to add, remove, edit all chart attributes in a tree-like structure. In order to add a new template, **Duplicate** an existing template, rename and edit its attributes.

Template properties and attribute editor

Every time a new chart is rendered in a web browser, it sends its **davLayout** / **template** value to the **DavService**. Depending on the value, it receives back the contents of one template and applies its values as a default attribute. The value of **davLayout** / **template** can be:

- `none` – no template will be applied
- template name – the requested template will be applied. E.g. `red` will return a template called 'red'.
- empty or not set – the template selected in **Default Template** property under **DavService** will be returned.

By default, the `classic` template is applied to all widgets. If **Default Template** property is changed, all widgets will be redrawn after the next page reload. Note, the template name can also be passed to PX file via hyperlink's Attribute Parameters.

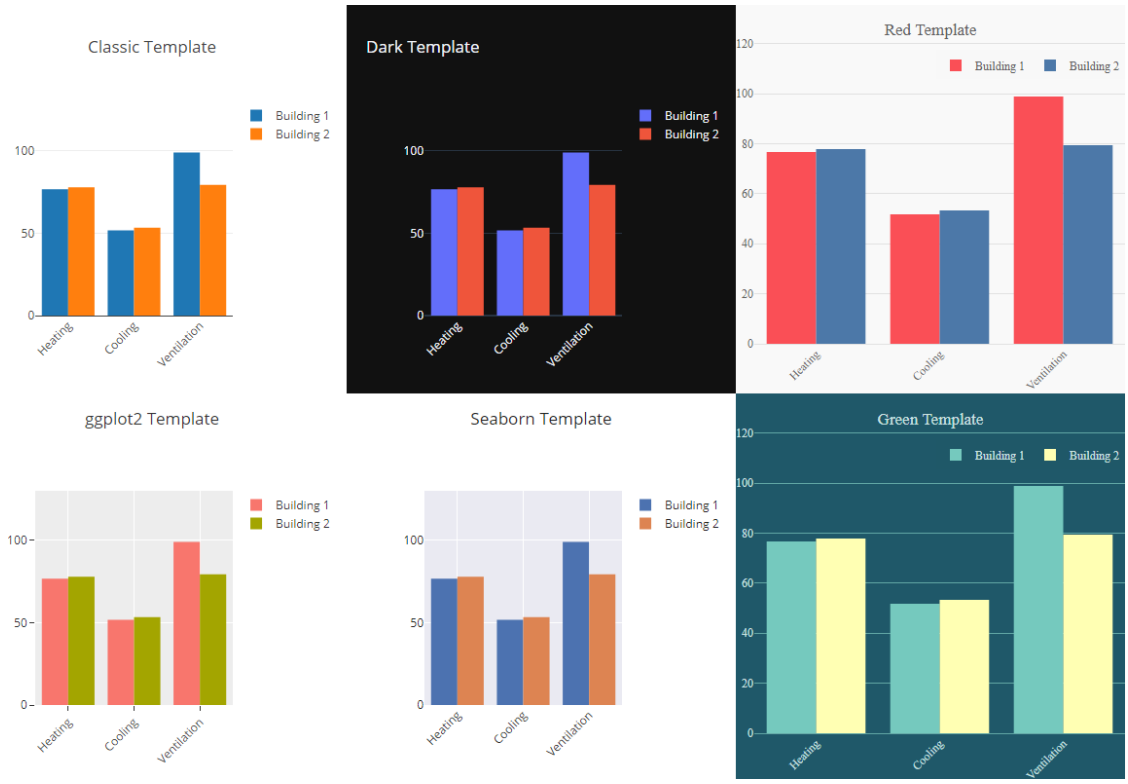


Figure 11: Template examples

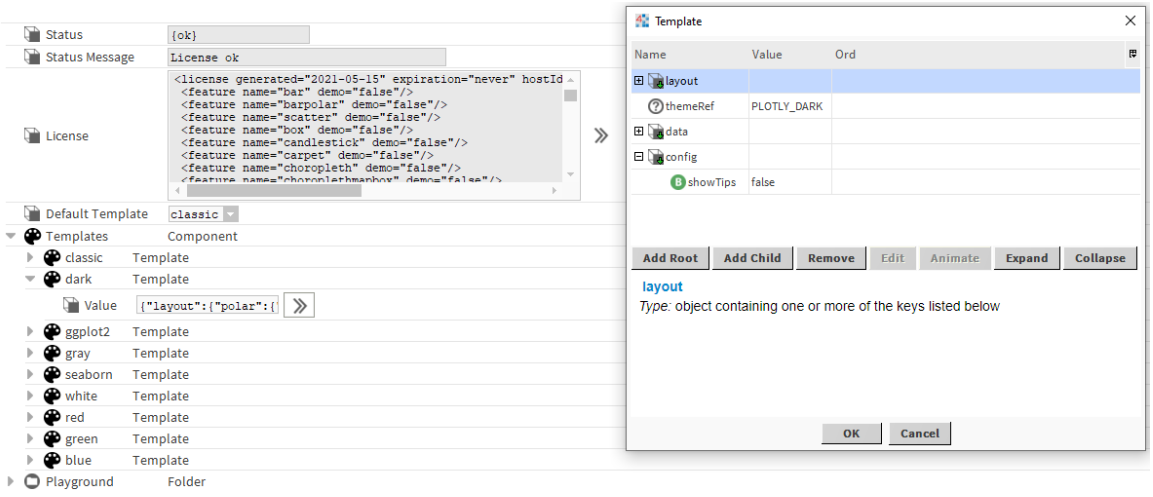


Figure 12: Template properties

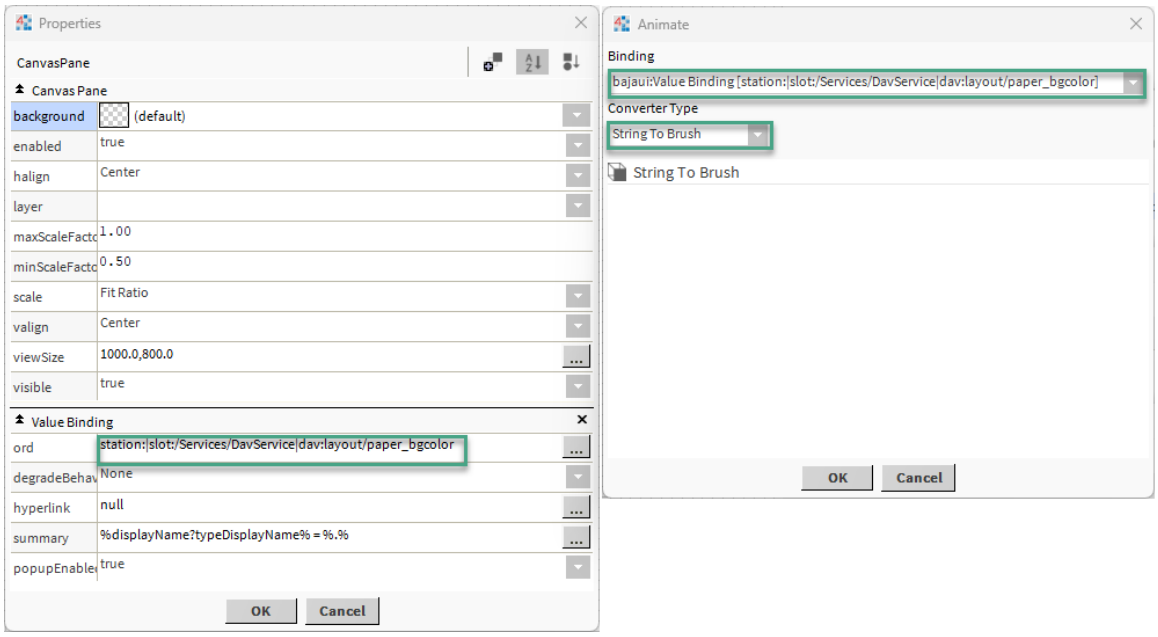


Figure 13: DAV Ord Scheme

2.7 DAV Ord Scheme

DAV library ord scheme **dav:** allows to use default template attributes in a regular Niagara bindings in order to change PX background when the template get changed. The ord `station:|slot:/Services/DavService|dav:layout/paper_bgcolor` and converter **StringToBrush** will update the background color every time **layout.paper__bgcolor** is changed in the default template.

2.8 Relativization and Parameterization

Niagara allows to create one PX file and then reuse it to see various sets of data. There are a few ways how that could be accomplished for Baudrate charts.

2.8.1 Relativization

Data ORDs, used for attribute animation, can be relative or absolute. The absolute path ensures that the data always resolves to a single unique component (`slot:/FCU1/HeatValve`) in the location that is specified by the ORD, regardless of where the PX file or the parent component is located. If the same PX file is attached to a view that belongs to a different component, the ORD will still resolve to the original component (`HeatValve`) because of the absolute path. However, if you make data binding relative (`slot:HeatValve`), then the path will resolve relative to its current parent ORD. This relative path makes the PX file resolve data bindings correctly to identically named components that reside in different locations, thus making one PX file usable in many views. So for parents `slot:/FCU1`, `slot:/FCU2`, and `slot:/Floor2/FCU20`, this binding will resolve `slot:/FCU1/HeatValve`, `slot:/FCU2/HeatValve`, and `slot:/Floor2/FCU20/HeatValve`.

To relativize PX file means to translate absolute bindings to relative ones. In order to automatically relativize ords, first create widget with absolute ords, add PX file as a view to a parent component it shall display, select the widget and choose **Relativize Ords** button in DAV Copilot toolbar. A

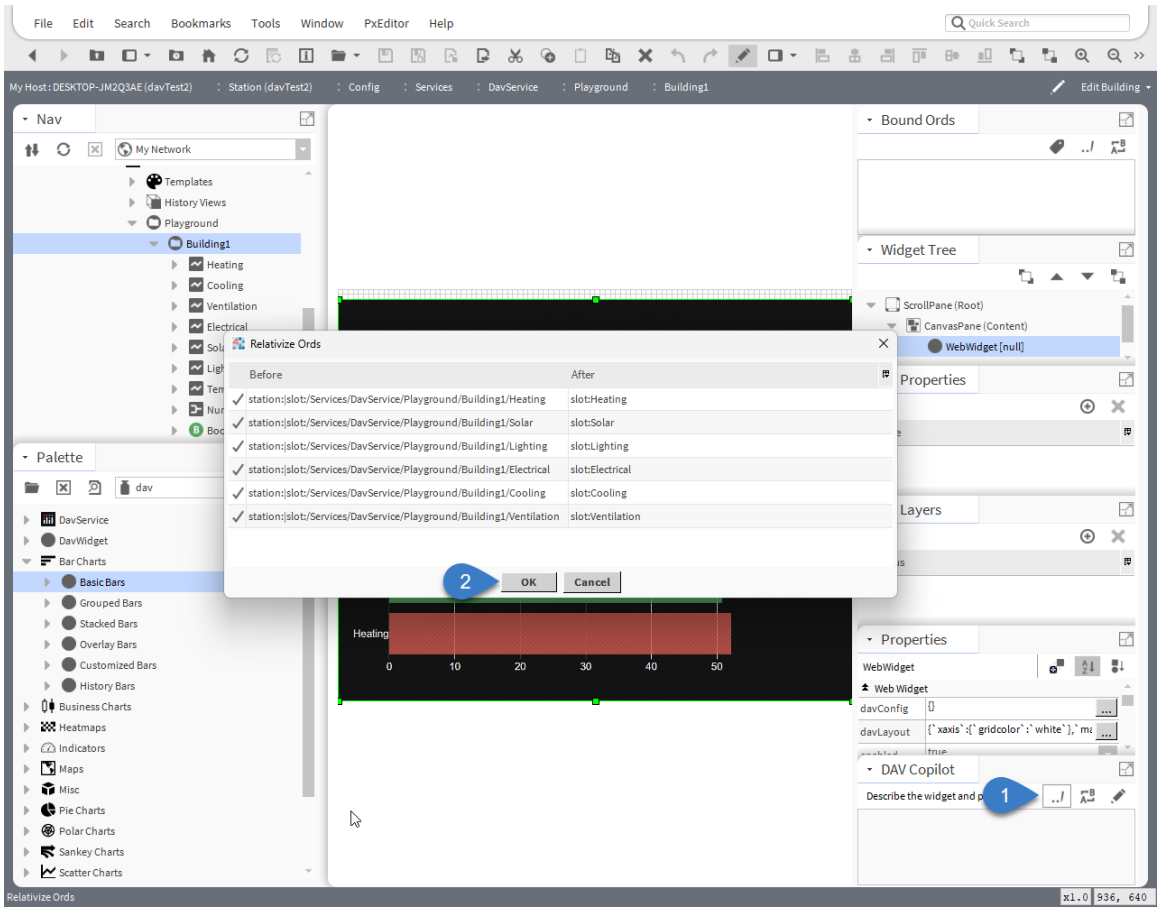


Figure 14: DAV Toolbar

dialog box will show all ords, that will be changed. After that the PX file can be used in all similar components.

2.8.2 ORD Parameters

Another way to reuse one PX file to visualize multiple components is to use ORD parameters in hyperlinks. Any part of ORD could be replaced with `$(var)` parameter, where `var` is a parameter name, e.g. `$(ord1)`, `$(ordA)`, `$(ahu)`. Then the value for these parameters could be passed in a hyperlink.

For example, ORD `history:/demo/$(ord)?timeRange=yesterday` becomes

`history:/demo/HeatValve?timeRange=yesterday`

if the hyperlink to the view contains parameter `ord = HeatValve`. Every PX file can have multiple ORD parameters.

Note, there is a special parameter case: `$(ord)`. It can be used as described above, but also serves as a placeholder for an ORD, which is added when the chart is added by drag & dropping points into PX file and selecting charts in palette via Make Widget menu. The `$(ord)` parameter will be replaced with an absolute ord of the selected component.

2.8.3 Attribute Parameters

Hyperlink parameters can also be used to set chart attribute values in layout, config and trace object. For example **layout.template** = `blue` parameter changes the name of template in all widgets on a linked PX screen, so all charts on the screen will be displayed with `blue` template, independently of the template set in **DavService / Default Template** property.

2.9 Machine Learning

Baudrate library includes multiple machine learning (ML) models to produce advanced analytical representations: trendlines and forecasts. They are fully calculated in browser, so no configuration shall be done on backend (station). All ML models are embedded into widgets using **postprocessing** attribute in **trace** object. This attribute can contain one or multiple lines in JavaScript to fit the model and calculate points to visualize it.

2.9.1 Trendlines

Trendlines are useful tool to calculate analytical dependency between variables.

A good example is how energy consumption in a building depends on degree days – a metric of how much (degrees) and for how long (days) the outside temperature sits below a certain level. To prepare data for scatter chart with trendline, one can use Series Transform component to merge energy and degree days histories and bind them to scatter chart attributes, so **x** axis represents degree days and **y** axis represent energy.

Then add another scatter trace and bind it to the same SeriesTransform component. Add **postprocessing** attribute to the **trace** and enter `[%trendX%, %trendY%, %trendF%] = Trendline.fit_predict(%degreedays%, %energy%, 'poly', 2)`. This function calculates polynomial trendline of 2nd degree using `degreedays` column as independent variable and `energy` column as dependent variable. The function returns three sets of values: trendline X coordinates, trendline Y coordinates and trendline formula. In this example these values are saved in `%trendX%`, `%trendY%`, and `%trendF%` variables in the history, so now one can draw trendline by binding `%trendX%` to **x** attribute, and `%trendY%` to **y** attribute using Format field.

Types of trendlines:

- `linear` – linear curve Y = a * X + b
- `poly` – polynomial curve Y = a1 * X + a2 * X^2 + a3 * X^3 + ... + b
- `power` – power curve Y = a * X^b
- `exponent` – exponential Y = a * e^(X * b)

2.9.2 Forecasts

Forecasts are used to predict future values of time series, i.e. timestamped historical data. The model used for forecasting works well with periodical data: outdoor temperature for the next week, energy consumption for the next year, or outdoor light level for the next day.

The model predicts one future period and requires at least two full periods of data and the number of data samples per period. The more periods of data are supplied, the better will be the forecast. The number of data samples should be calculated as the number of historical values for one repetitive cycle.

For example, if we collect 15 minute history for energy consumption, and we need to forecast consumption for tomorrow, then one cycle is 24 hours and the number of samples is 24 * 60 / 15

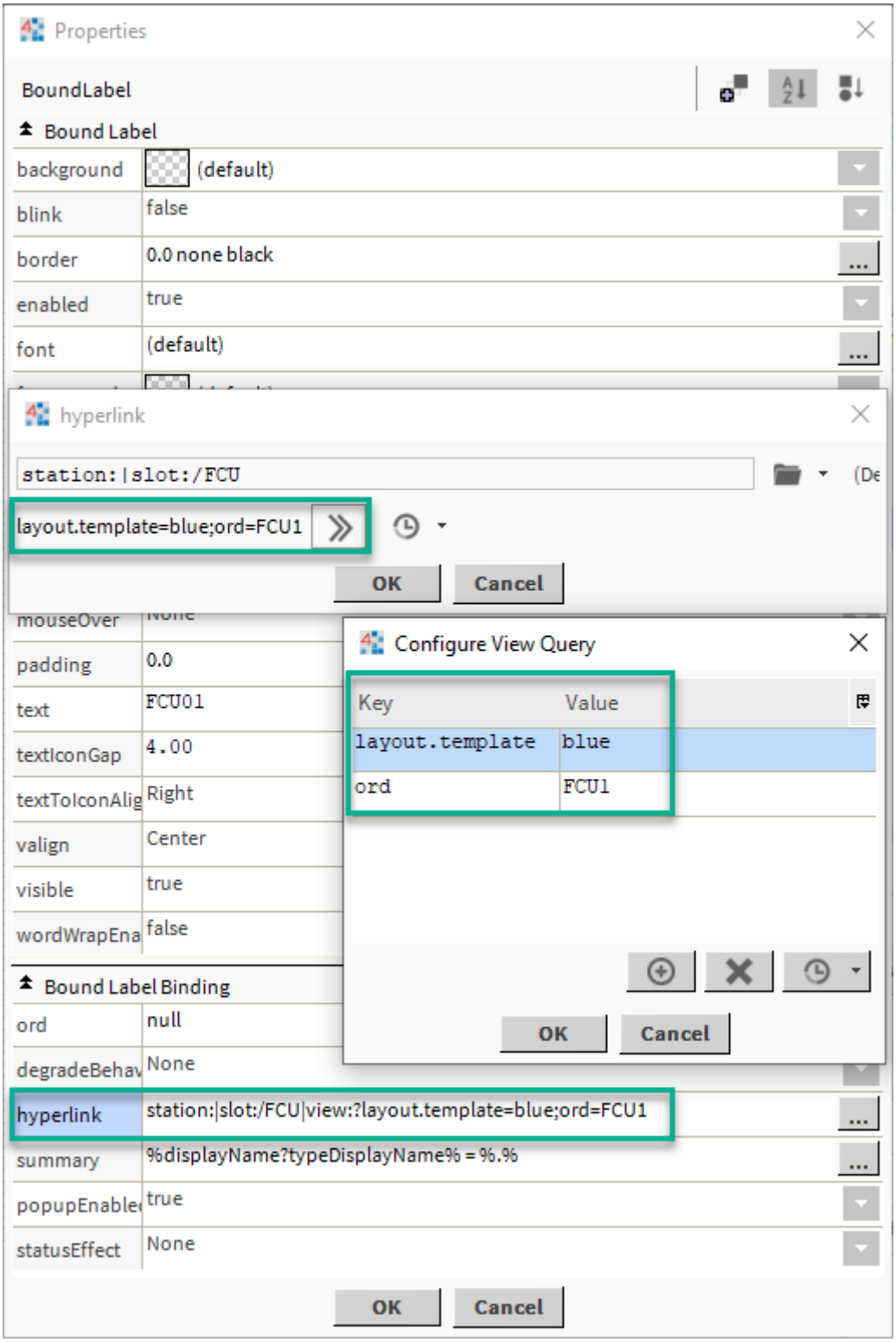


Figure 15: Hyperlink parameters

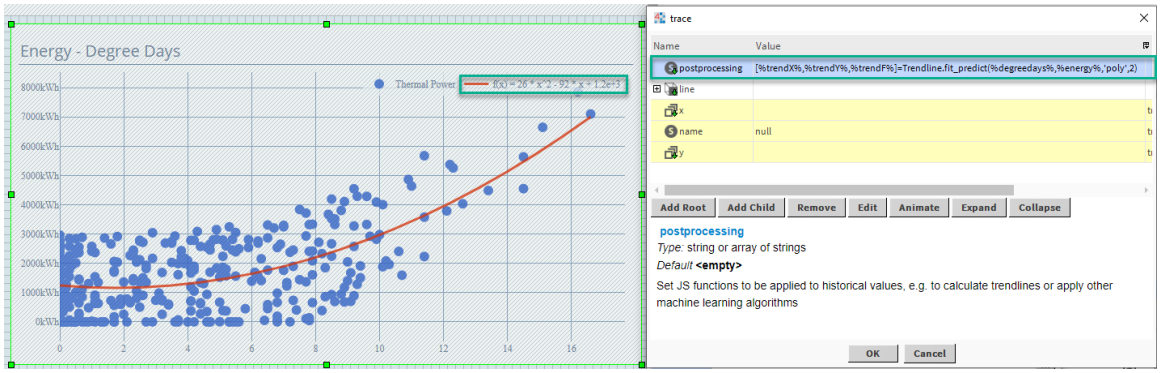


Figure 16: Trendline

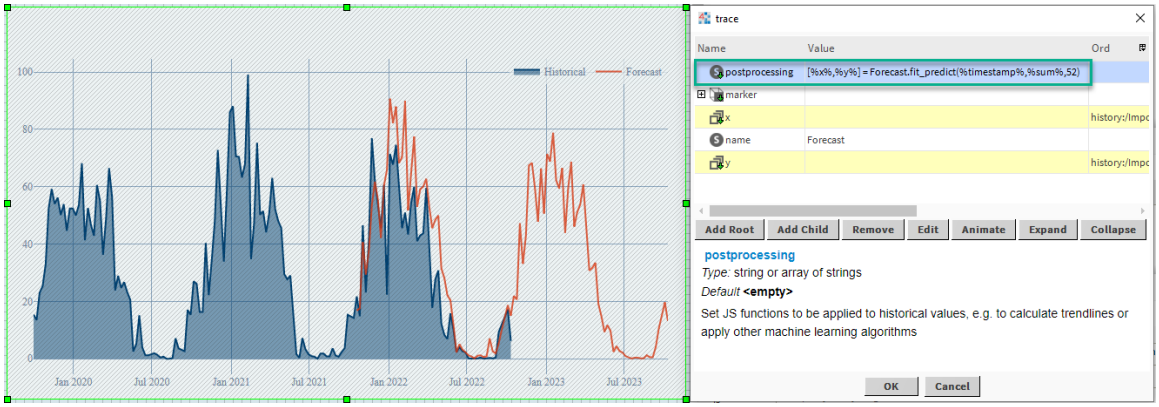


Figure 17: Forecast

= 96. In this case the model will match daily fluctuation of energy. If we need to predict yearly fluctuations, then we can use rollup function to reduce the amount of data to one per day (or one per week, one per month) and the number of samples will be 365 (days in a year). Note, the model requires at least two full cycles, so there shall be at least 365 * 2 = 730 samples available.

The chart configuration is done very similarly to trendlines, except the data shall be timestamped. The **postprocessing** attribute value should be equal to `[%x%, %y%] = Forecast.fit_predict(%timestamp%, %avg%, 52)`, where `timestamp` is history timestamp column, `avg` is history data column, `52` – number of samples per period, `x` and `y` – variable names for forecasted values. These values can be displayed on the chart by binding to **x** and **y** attribute using Format field.

2.9.3 DAV Copilot

The DAV Copilot is an AI-based assistant that helps generate charts from textual descriptions. It's an efficient tool for converting your ideas into visual representations with just a few keystrokes. To begin using the DAV Copilot, you'll need to subscribe to an OpenAI paid plan and obtain an OpenAI API key. Follow these steps to set up your DAV Copilot:

1. **Sign up for OpenAI's paid plan**: Visit OpenAI's subscription page and follow the prompts to complete your subscription.
2. **Obtain your OpenAI API Key**: After signing up, generate your API key. The OpenAI Dashboard will guide you through this process. Remember to keep this key confidential and secure.

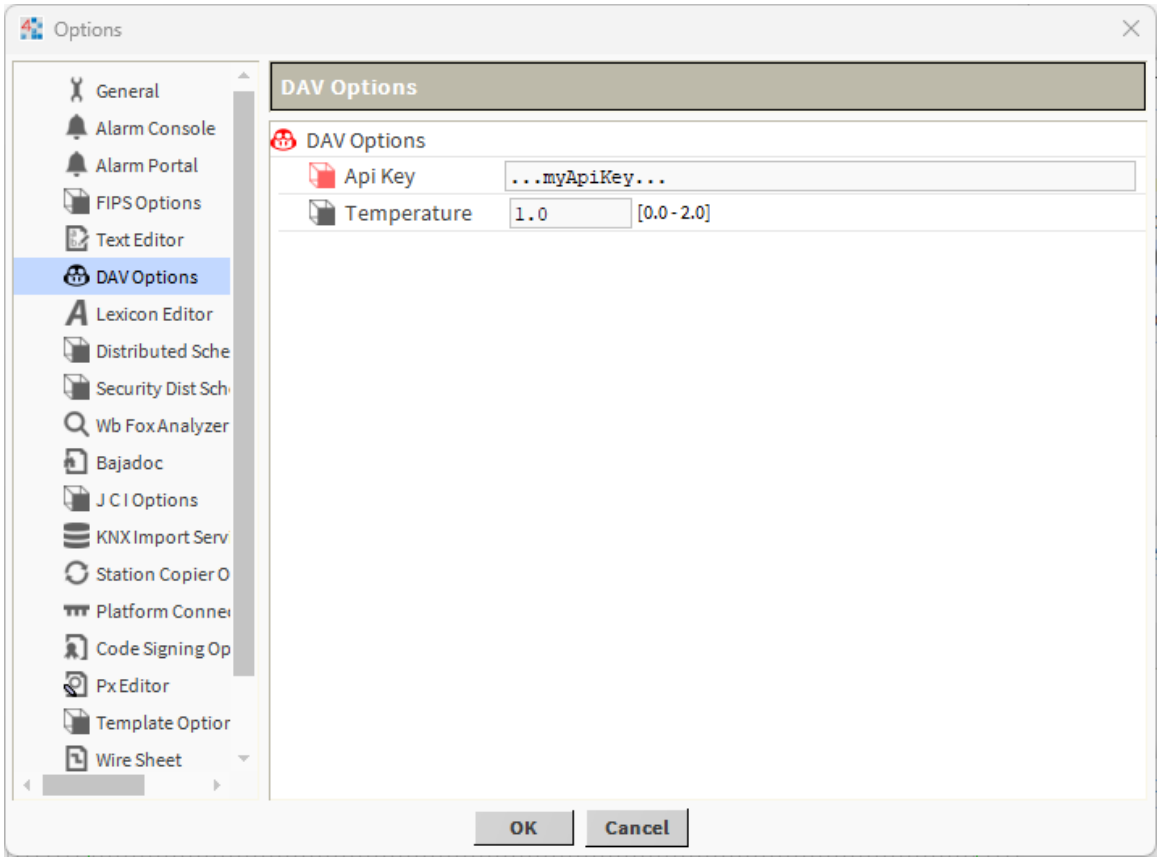


Figure 18: DAV Copilot

3. **Input your API Key in the DAV Copilot**: After obtaining your API key, open your Workbench application. Navigate to the top menu and select `Tools > Options > DAV Options`. Enter your OpenAI API key in the 'API Key' field.

Your DAV Copilot setup is now complete!

Here's a simple guide to get you started:

1. Begin by opening your PX file.
2. In the text input area, describe the widget you'd like to create. For instance, type "Create a pie chart for weekly energy consumption," and then press Enter.
3. The DAV Copilot will generate the corresponding chart using its AI capabilities and add it to your PX file automatically.

If you need to modify an existing chart, follow these steps:

1. Select the chart you want to modify.
2. Enter your modification instructions in the DAV Copilot menu.
3. The DAV Copilot will update the chart based on your text description.

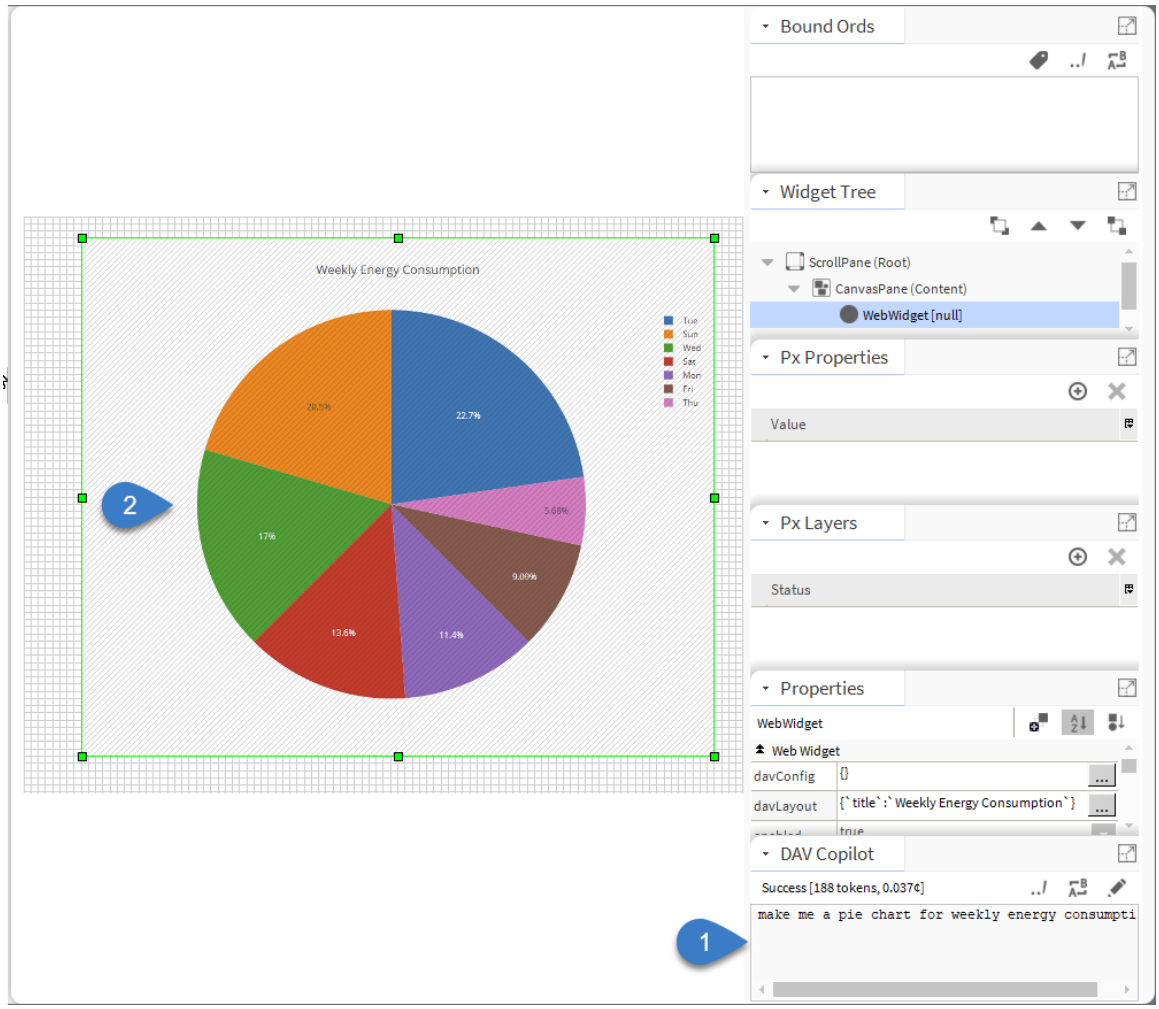


Figure 19: DAV Copilot

Figure 20: Plotly Chart Studio

2.10 Additional resources

As the library is based on a popular plotly.js charts, there is a lot of online resources available to answer questions and get inspiration of how to build good-looking charts. Here we describe some of them.

2.10.1 Plotly.com

Plotly.com website provides tons of examples and in-depth descriptions of various plotly.js aspects. Almost everything shown there could be implemented in Baudrate library.

2.10.2 Plotly Chart Studio

Plotly Chart Studio is an online repository of thousands of plotly.js charts, searchable by keywords, chart type, popularity. Every chart can be viewed and edited in online chart editor (see its user guide), the great tool to see how various attribute values affect the chart.

Every chart in Plotly Chart Studio can be exported into file (press **Export** button and select **json** format) and then imported into Niagara (select **Import JSON** in the DAV Copilot toolbar and find the exported file). The chart appears in Niagara exactly as seen online, the only thing left is to bind real-time or historical data to it.

2.10.3 Plotly.js JSON Editor

Another Plotly.js JSON Editor also allows to see and edit multiple chart examples.

2.10.4 Search Engines

Due to plotly.js popularity it is often very easy to find an answer to chart attribute related questions online. For example, googling “plotly.js scatter how to change grid color” returns multiple relevant

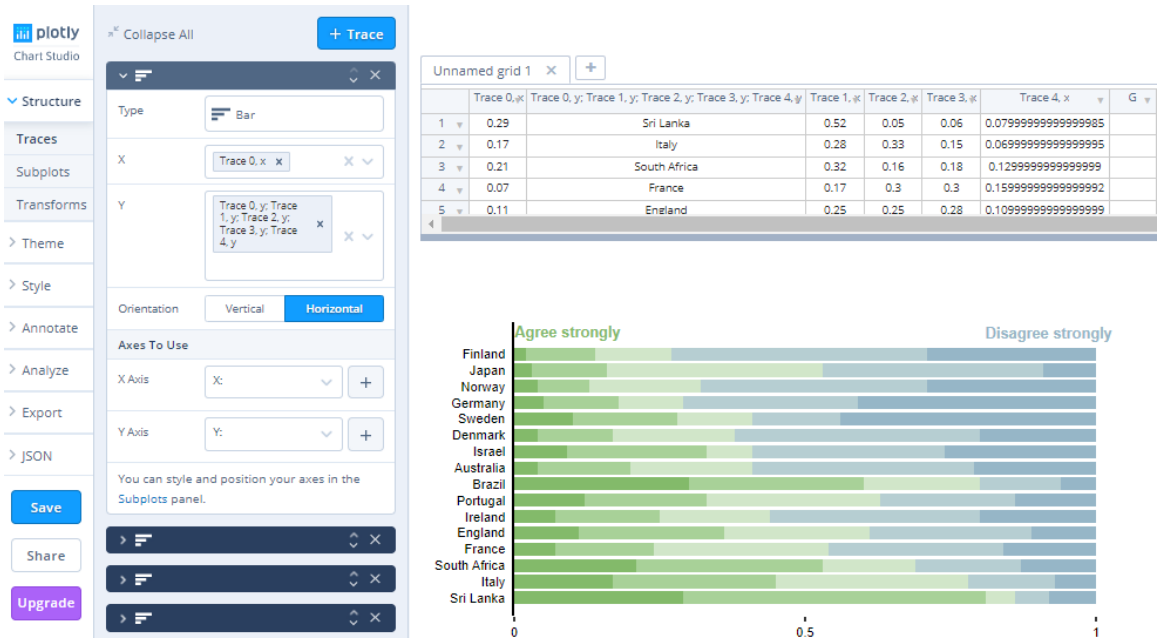


Figure 21: Plotly Chart Editor

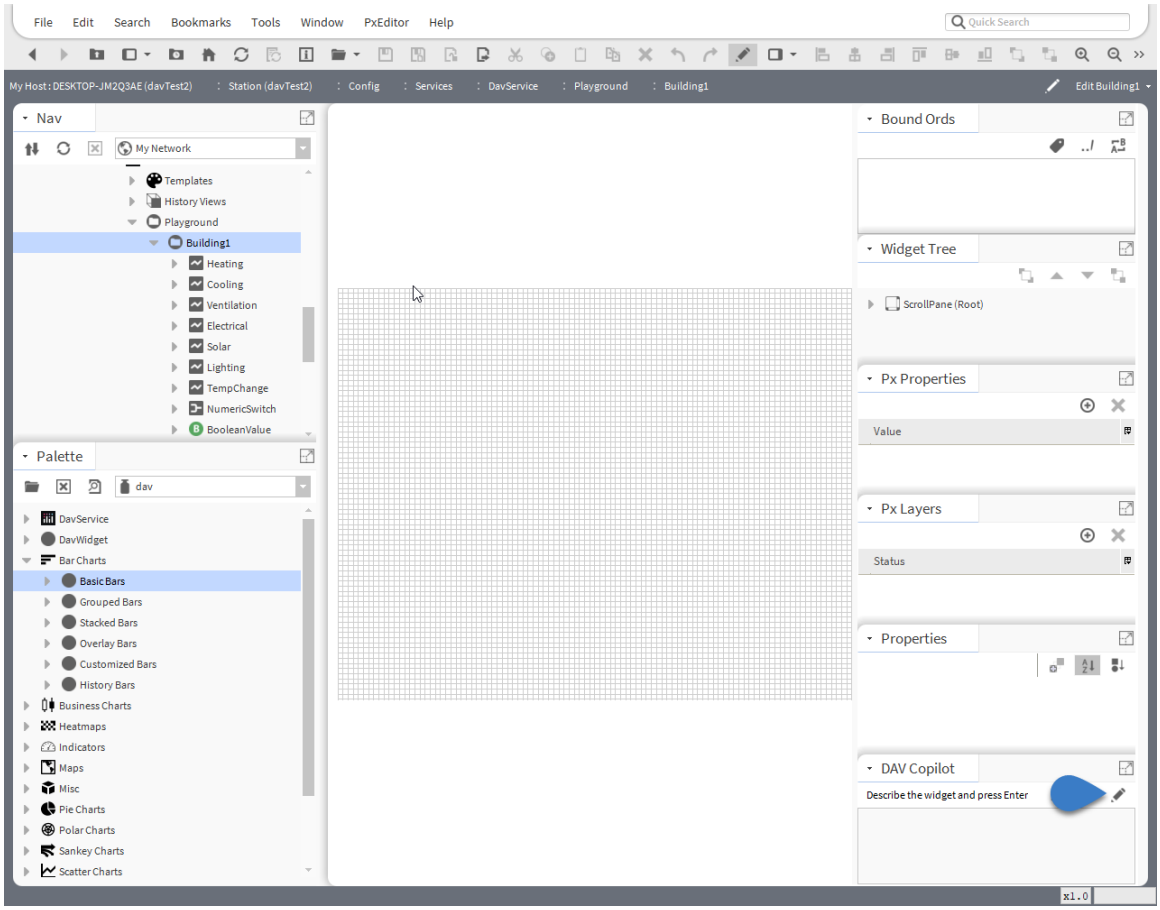


Figure 22: JSON import

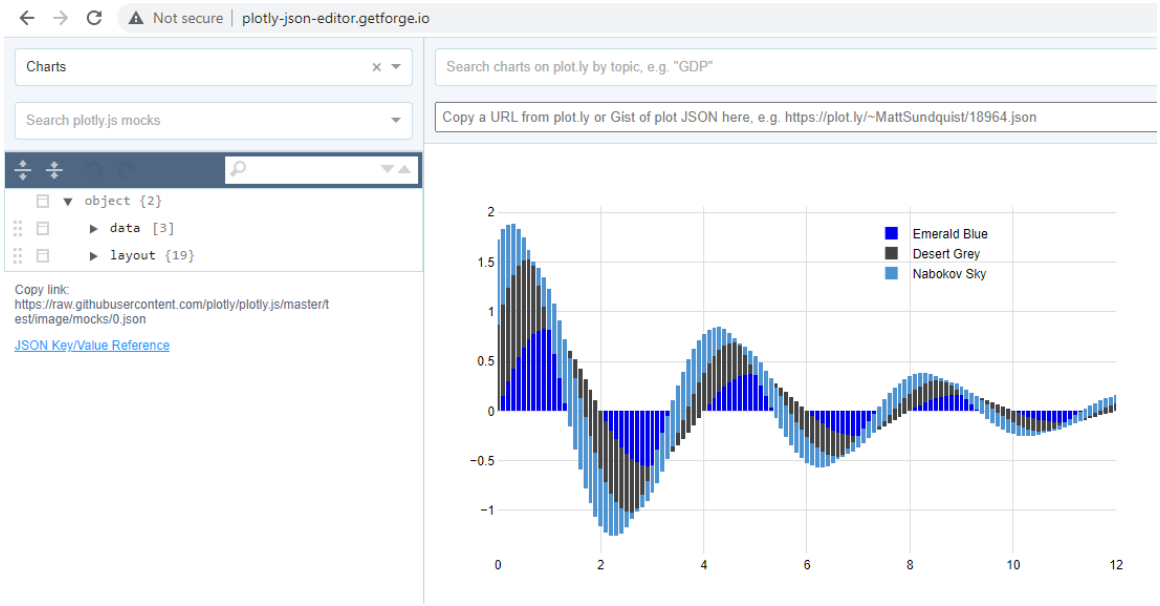


Figure 23: Plotly alternative editor

links to community.plotly.com and stackoverflow.com, providing **layout.yaxis.gridcolor** attribute as the right answer. It can be modified under **davLayout** property.

2.10.5 Single-Page Reference

Plotly.js Single-Page Reference is a list describing all plotly.js attributes for every type of chart.

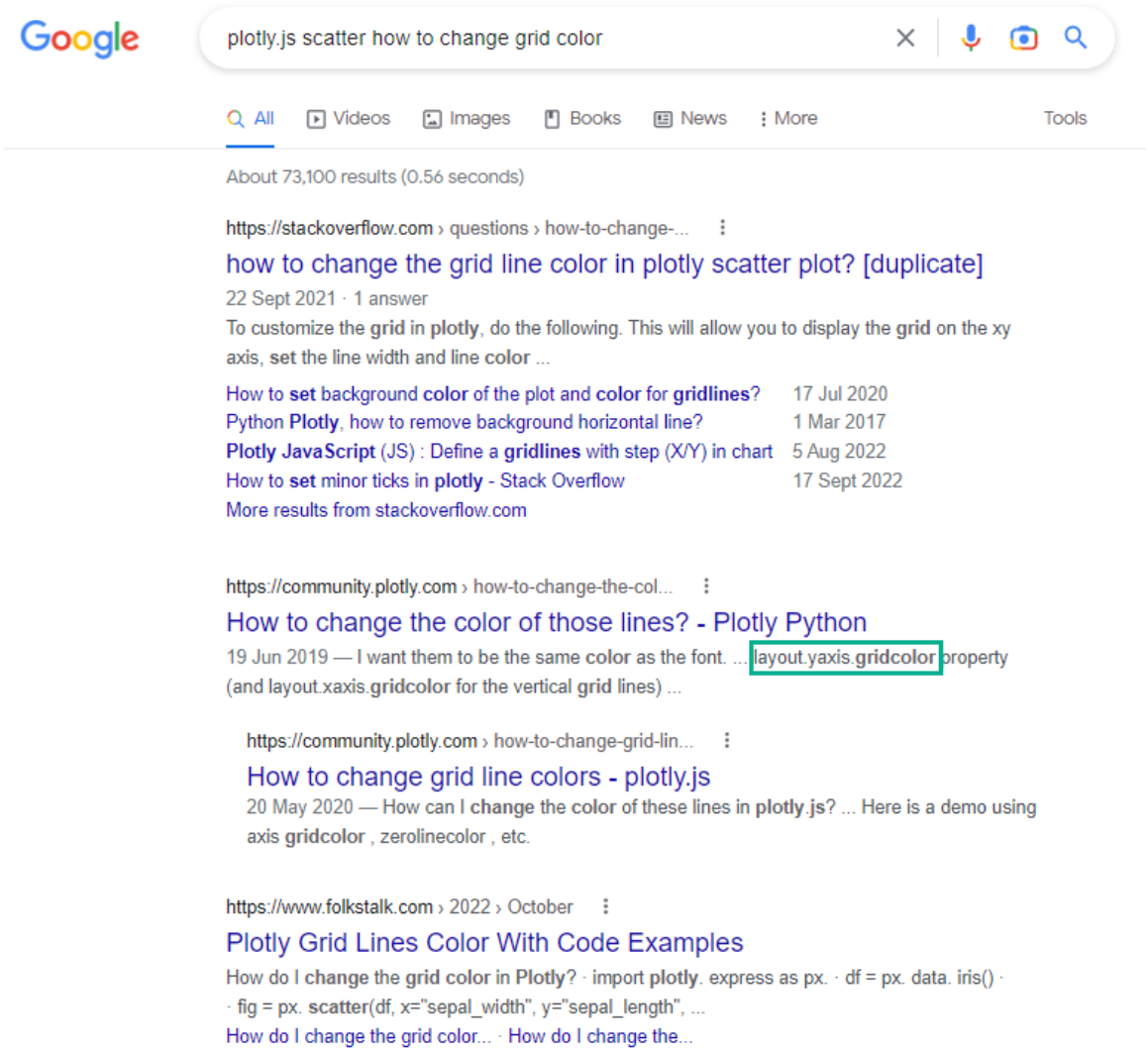


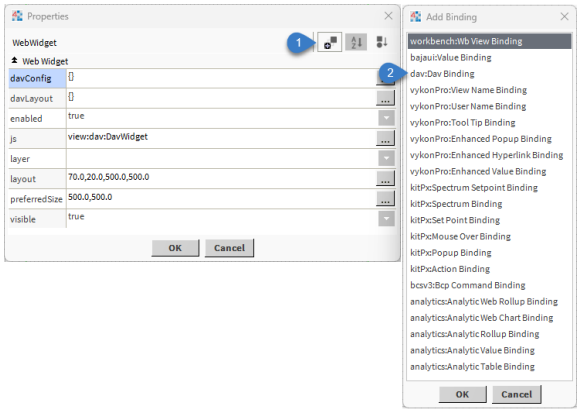
Figure 24: Google results

3 Step-by-step Guide

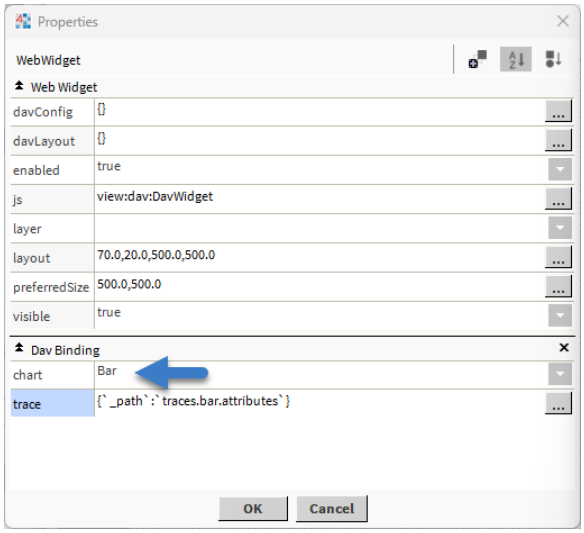
3.1 Bar chart animated with numeric points

In this example we will build a simple bar chart and animate it using numeric points.

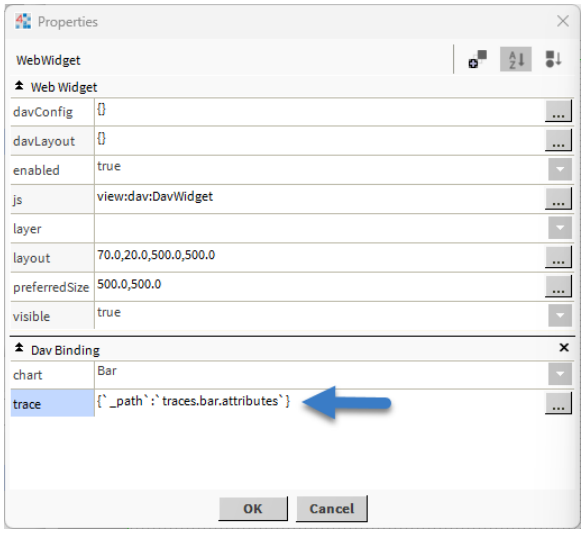
1. Drag & drop **EmptyWidget** from the DAV palette to the px file.
2. Click on **Add Binding** button and select **Dav Binding** in the dialog box. Click **OK**.



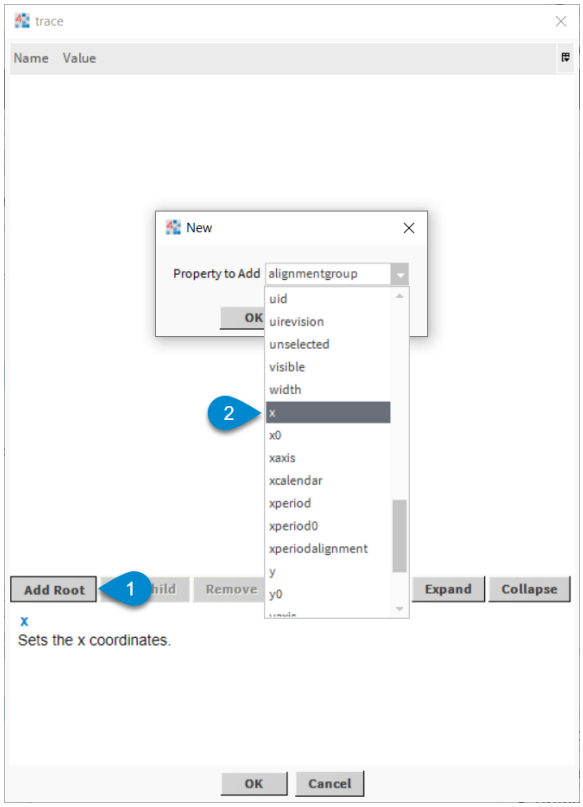
3. Select **Bar** chart type in **chart** property.



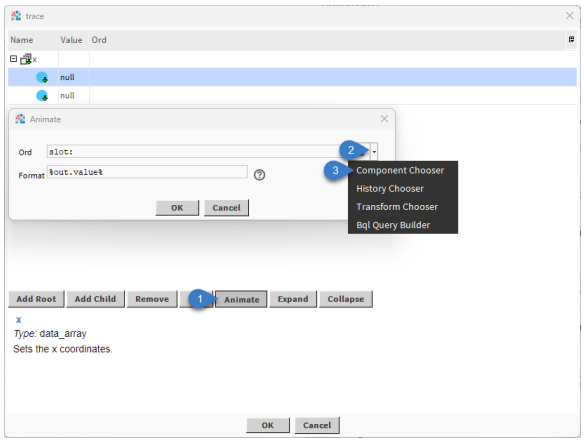
4. Press on **trace** property to open a dialog box. This property might contain a very complex tree-like set of attributes.



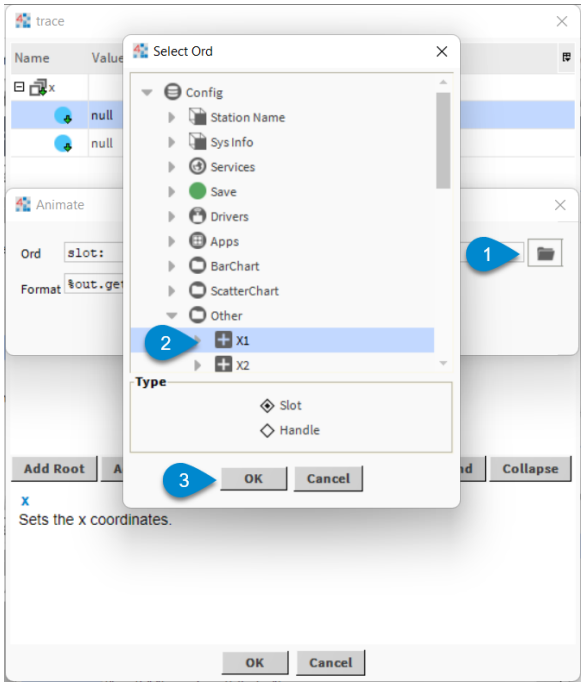
5. Click **Add Roots** and add **x** axis attribute.



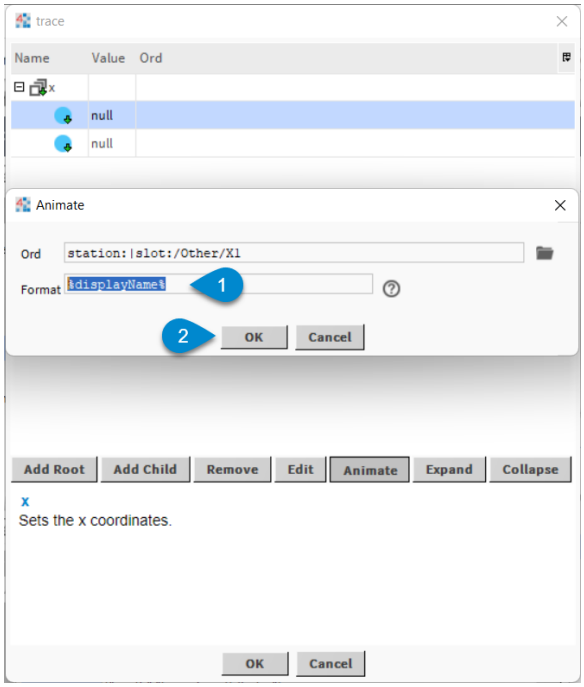
6. Click **Add Child** to add a number of points you are looking to add to the chart.
7. While child attribute is selected click **Animate** button. From drop-down menu select and click **Component Chooser**.



8. Click on dialog box and select a numeric point in Niagara station. Click **OK**.



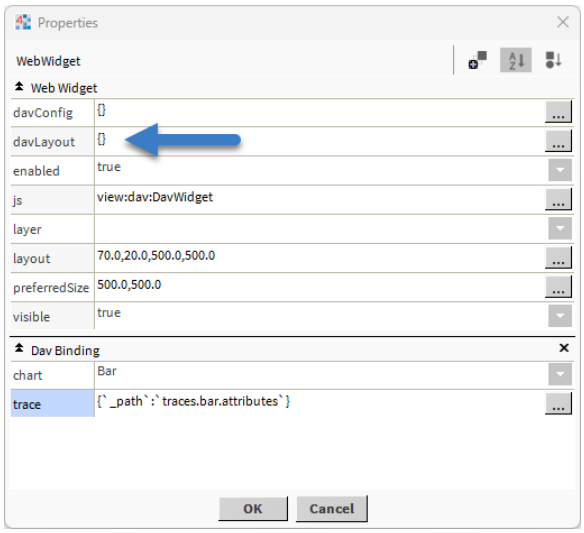
9. Enter **%displayName%** in **Format** field to display point display names on **x** axis. Click **OK**.



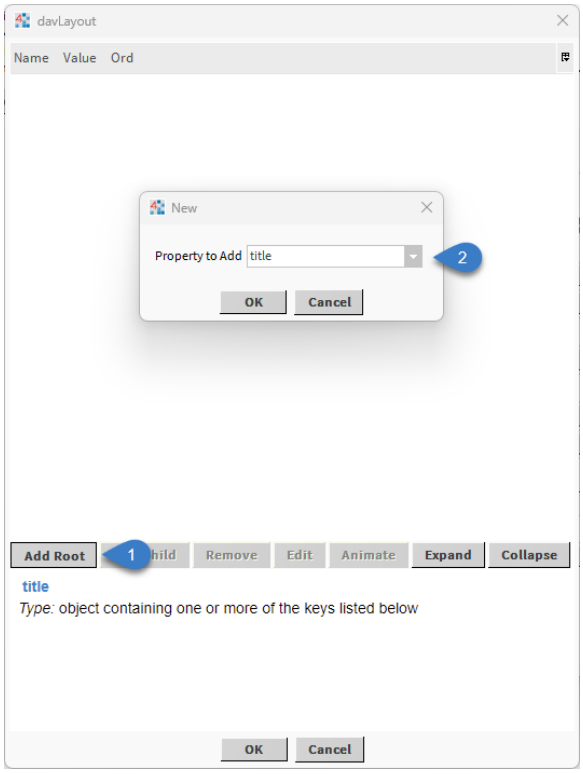
10. Click **Add Roots** and add **y** axis attribute.
11. By clicking **Add Child** add the necessary number of points you are looking to add to the chart.
12. While child attribute is selected click **Animate** button.
13. Click on dialog box and select numeric point in Niagara station. Click **OK**.
14. Enter **%out.value%** to display point **out** property on **y** axis. Click **OK**.

NOTE: for all attributes that are currently animated the background color will be changed to yellow.

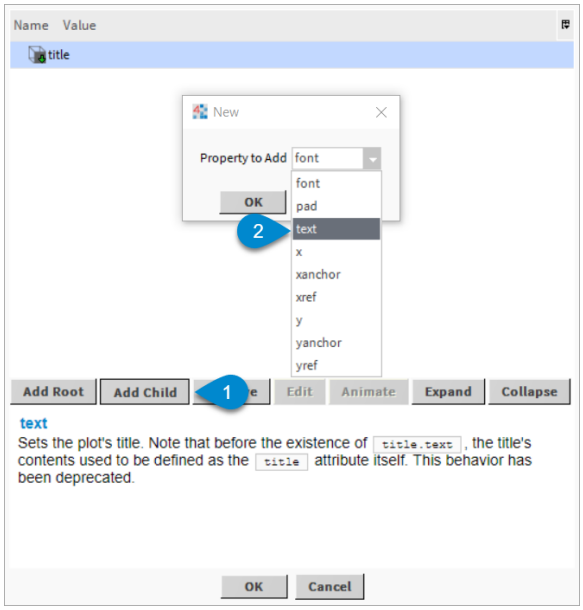
16. Click **OK** to close **trace** property dialog box.
17. Click **davLayout** to open the dialog box.



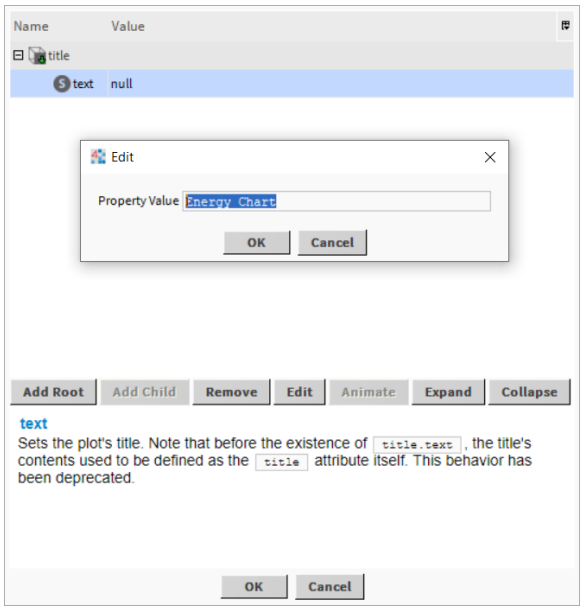
17. Click **Add Roots** and add **title** attribute.



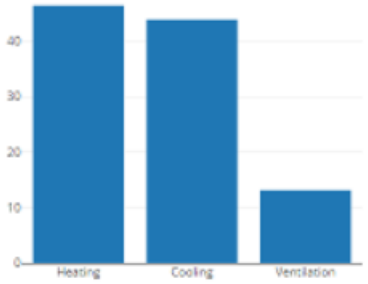
18. Select **title** attribute and click **Add Child** and add **text** attribute.



19. Double-click **text** attribute and enter the desired chart name.



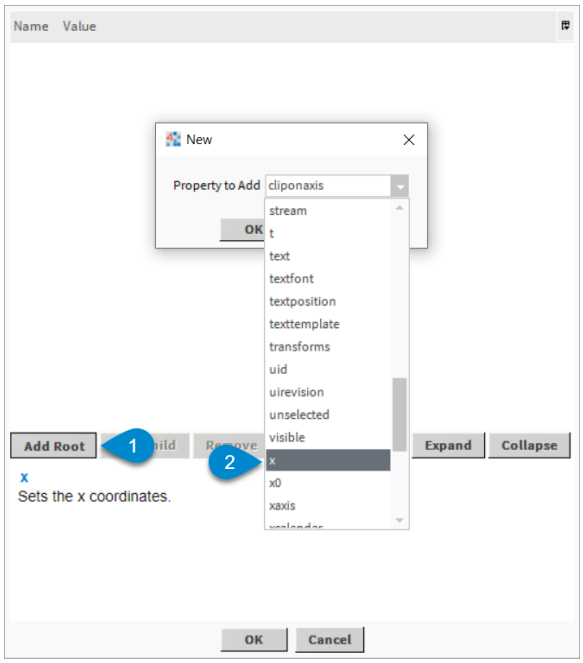
20. Click **OK** to close **davLayout** property dialog box.
21. Press **OK** to close the widget and see the result.



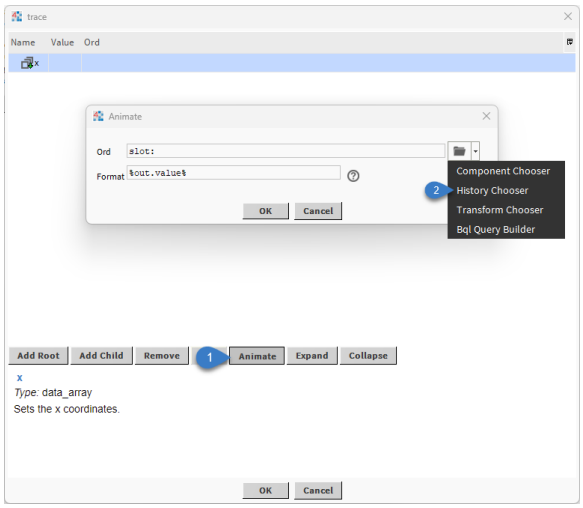
3.2 Scatter chart animated with numeric history

In this example we will build a simple scatter chart.

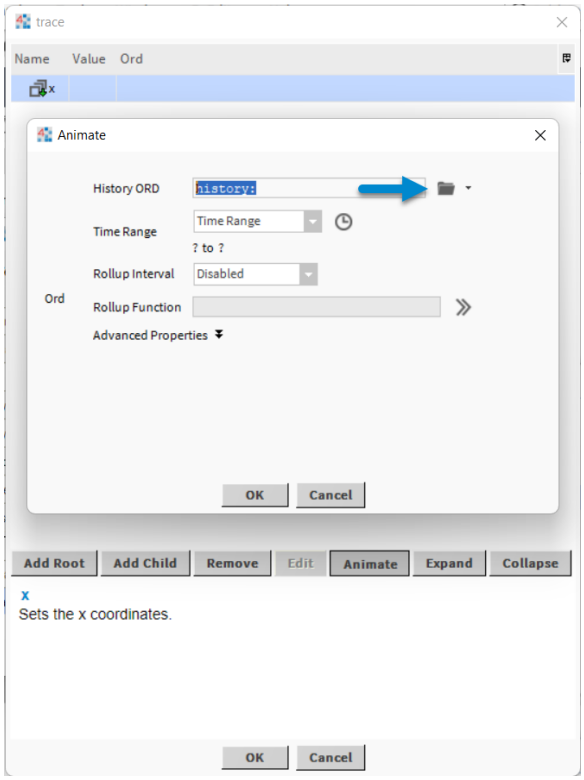
1. Drag and drop **EmptyWidget** from the **dav** palette to the px file.
2. Click on **Add Binding** button and select **Dav Binding** in the dialog box. Click **OK**.
3. Select **Scatter** chart type in **chart** property.
4. Press on **trace** property to open the dialog box.
5. Click **Add Root** and add **x** axis attribute.



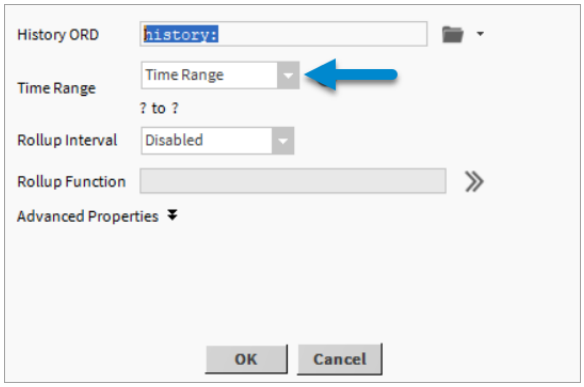
6. While **x** attribute is selected click **Animate** button. Enter %timestamp% in Format field to display history timestamps on x axis. From drop-down menu select and click **History Chooser**.



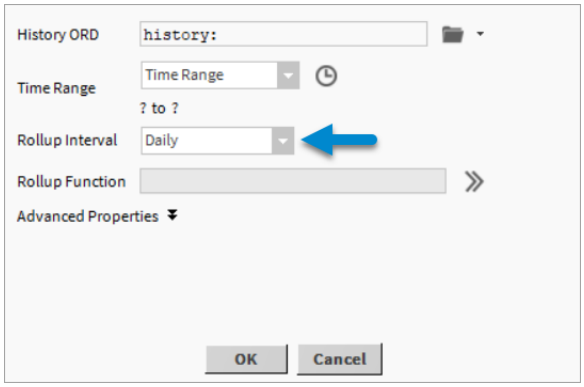
7. Click **History Ord Chooser** button and select history to be bound.



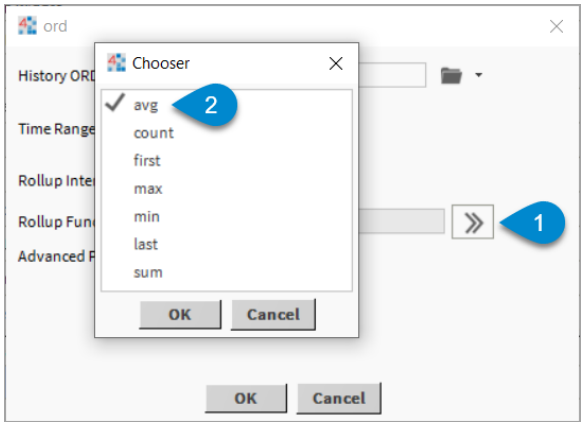
8. Select **Time Range** without specifying the period, that will show the full history length.



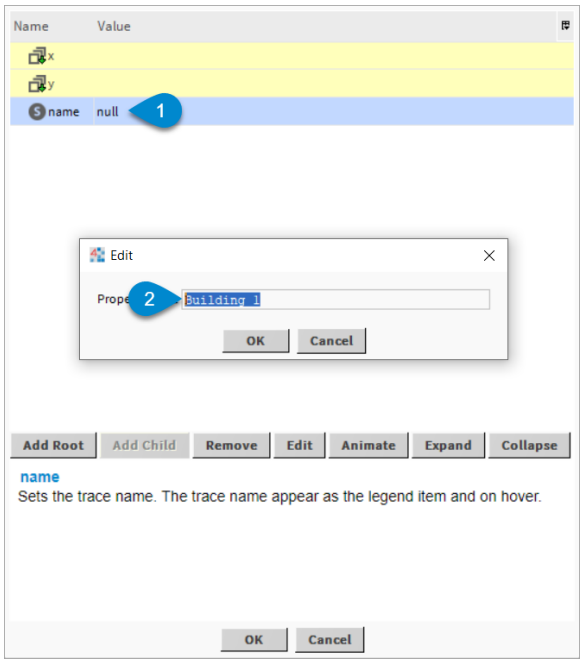
9. Optional Item. In order to roll up the history values select the desired **Rollup Interval**. For example if you have annual history you might want to select **Daily** interval.



10. If **Rollup Interval** is used click on **Rollup Function** and select the necessary function. For this example let's select **avg**. Click **OK**. Click **OK** to exit a dialog box.

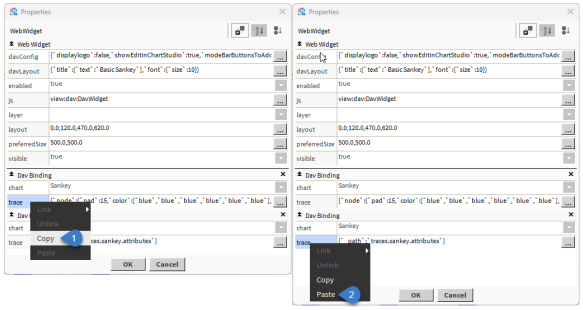


11. Click **Add Root** and add **y** axis attribute.
12. While **y** attribute is selected click **Animate** button to open the dialog box. Enter **%avg%** in **Format** field (if you have not used rollup function enter "%value%") to display history values on **y** axis. Repeats steps 7-8.
13. Click **Add Root** and add **name** attribute.
14. Double-click on **name** attribute and enter a series name e.g. "Building 1".

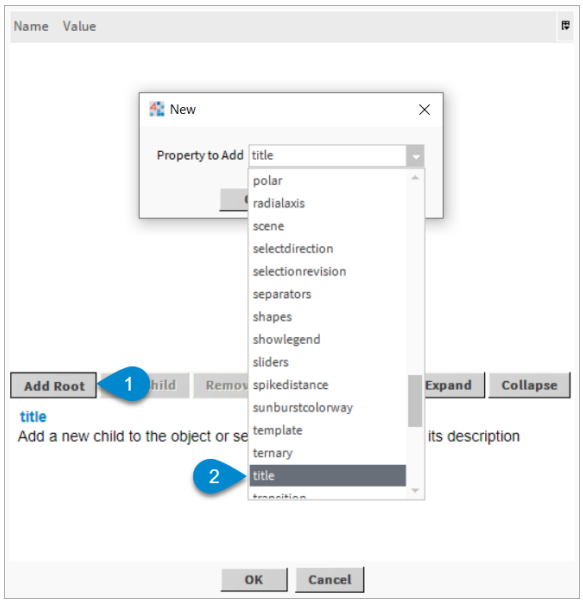


17. Click **OK** to close **trace** property dialog box.
18. Repeat steps 2 to 17 to add another series to the chart.

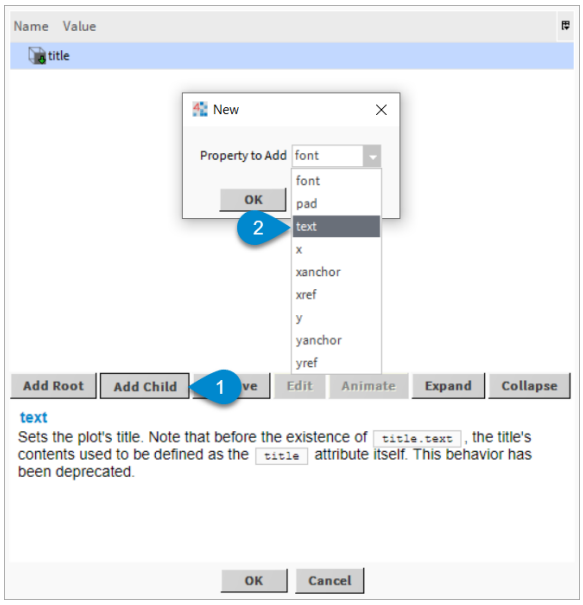
NOTE: You can copy any of the properties. For example, you can right-click and select copy for the existing **trace** properties, then select paste for a newly created **trace** property.



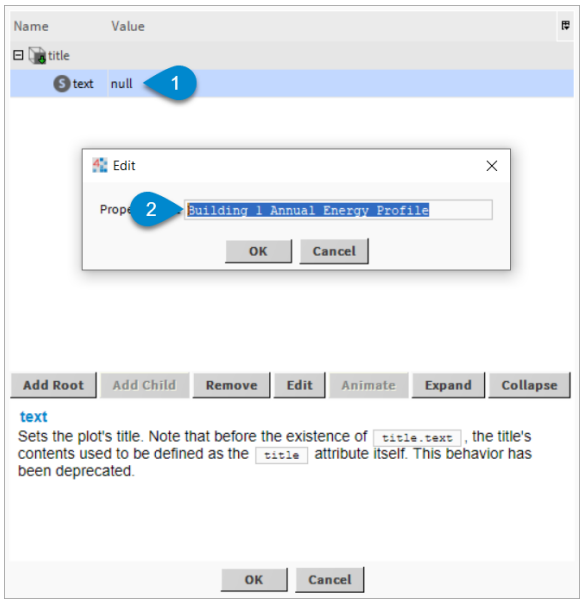
19. Click **davLayout** to open the dialog box.
20. Click **Add Root** and add **title** attribute.



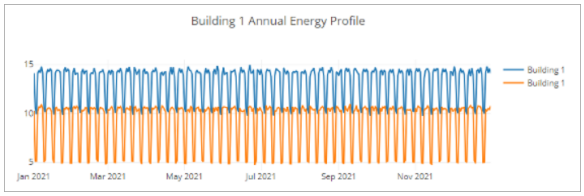
21. Select **title** attribute and click **Add Child** and add **text** attribute.



22. Double-click **text** attribute and enter the desired chart name.



23. Click **OK** to close **davLayout** property dialog box.
24. Press **OK** to close the widget and see the result.



4 Attribute Reference

4.1 Trace Attributes

The **trace** objects attributes control aspects of a chart itself: the colors, the grids, the data etc. The full list of attributes can be found Trace Reference online.

4.2 Layout Attributes

The **davLayout** object contains attributes that control positioning and configuration of non-data-related parts of the figure such as:

- Dimensions and margins, which define the bounds of paper coordinates
- Figure-wide defaults: fonts, colors, hover-label and modebar defaults
- Title and legend
- Color axes and associated color bars
- Subplots of various types on which can be drawn multiple traces: xaxis, yaxis, scene, ternary, polar, geo, mapbox subplots
- Non-data marks: annotations, shapes, images
- Controls which can trigger DAV library functions when interacted with by a user: updatemenus and sliders

4.2.1 Frequently Used Layout Attributes

- title.text - figure title.
- title.font.size - figure title size.
- title.font.color - figure title color.
- showlegend - enable/disable legend.
- margin - figure margins.
- barmode - determines how bars are displayed on the graph. With “stack”, the bars are stacked on top of one another. With “relative”, the bars are stacked on top of one another. With “group”, the bars are plotted next to one another centered around the shared location. With “overlay”, the bars are plotted over one another, you might need to an “opacity” to see multiple bars.
- colorscale - represent a mapping between the range 0 to 1 and some color domain within which colors are to be interpolated (unlike discrete color sequences which are never interpolated).
- colorway - sets the default trace colors.
- modebar - list of configurations for modebar.

Full list of plotly attributes can be found in Layout Reference online.

4.3 Config Attributes

The **config** object’s attributes control behaviours, which are not considered part of the figure itself, e.g. the behaviour of the `modebar`, or time range selector buttons, or how the figure reacts to mouse actions like scrolling.

Descriptions of config attributes can be found in Configuration Options online.

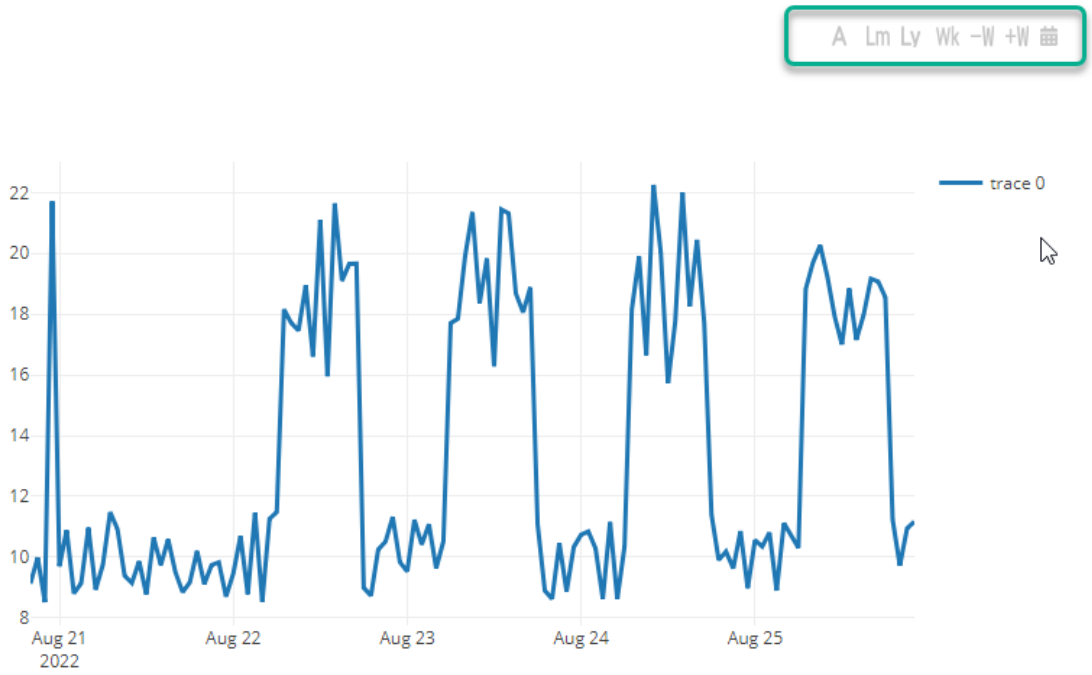


Figure 25: Time range buttons

4.3.1 Time Range Selector Buttons

Chart time range can be changed dynamically via buttons in the mode bar.

To display time range buttons add any combination of the following values as children to `config.modeBarButtonsToAdd` or as grandchildren to `config.modeBarButtons`:

- histAuto, histToday, histYesterday, histWeekToDate, histLastWeek, histMonthToDate, histLastMonth, histYearToDate, histLastYear – standard Niagara time selectors
- prevDay, prevWeek, prevMonth, prevYear – select previous day / week / month / year relative to the displayed period.
- nextDay, nextWeek, nextMonth, nextYear – select next day / week / month / year relative to the displayed period.

4.3.2 Frequently Used Config Attributes

- `plotlyServerURL` - use value `https://chart-studio.plotly.com` in order to export the chart to the chart studio.
- `modeBarButtonsToAdd.downloadJSon` - allows to export the chart as JSON file.
- `displayLogo` - to enable/disable plotly logo.