Technical Document

Niagara Series Transform Guide

May 24, 2022

niagara4

Niagara Series Transform Guide

Tridium, Inc.
3951 Westerre Parkway, Suite 350
Richmond, Virginia 23233
U.S.A.

Confidentiality

The information contained in this document is confidential information of Tridium, Inc., a Delaware corporation ("Tridium"). Such information and the software described herein, is furnished under a license agreement and may be used only in accordance with that agreement.

The information contained in this document is provided solely for use by Tridium employees, licensees, and system owners; and, except as permitted under the below copyright notice, is not to be released to, or reproduced for, anyone else.

While every effort has been made to assure the accuracy of this document, Tridium is not responsible for damages of any kind, including without limitation consequential damages, arising from the application of the information contained herein. Information and specifications published here are current as of the date of this publication and are subject to change without notice. The latest product specifications can be found by contacting our corporate headquarters, Richmond, Virginia.

Trademark notice

BACnet and ASHRAE are registered trademarks of American Society of Heating, Refrigerating and Air-Conditioning Engineers. Microsoft, Excel, Internet Explorer, Windows, Windows Vista, Windows Server, and SQL Server are registered trademarks of Microsoft Corporation. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Mozilla and Firefox are trademarks of the Mozilla Foundation. Echelon, LON, LonMark, LonTalk, and LonWorks are registered trademarks of Echelon Corporation. Tridium, JACE, Niagara Framework, and Sedona Framework are registered trademarks, and Workbench are trademarks of Tridium Inc. All other product names and services mentioned in this publication that are known to be trademarks, registered trademarks, or service marks are the property of their respective owners.

Copyright and patent notice

This document may be copied by parties who are authorized to distribute Tridium products in connection with distribution of those products, subject to the contracts that authorize such distribution. It may not otherwise, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior written consent from Tridium, Inc.

Copyright © 2022 Tridium, Inc. All rights reserved.

The product(s) described herein may be covered by one or more U.S. or foreign patents of Tridium.

Contents

About this guide

This topic contains important information about the purpose, content, context, and intended audience for this document.

Product documentation

This document is part of the Niagara technical documentation library. Released versions of Niagara software include a complete collection of technical information that is provided in both online help and PDF format. The information in this document is written primarily for Systems Integrators. To make the most of the information in this book, readers should have some training or previous experience with Niagara software, as well as experience working with JACE network controllers.

Document content

Document change log

Updates (changes and additions) to this guide are listed below.

May 24, 2022

Replaced AX images with N4 images.

Added "Cleanser Node" topic to the "Series Transformations", "Components" and "windows" chapter.

November 5, 2013

NiagaraAX-3.8 release revision. Added sections describing the TimeShift and the FilterNode components.

August 28, 2012

NiagaraAX-3.7 release revision.

Related documentation

Several other documents are available for learning how to use the Niagara McQuay driver.

- *Niagara Drivers Guide* explains concepts.
- *Getting Started with Niagara* explains concepts.

Chapter 1 Series transformations

Topics covered in this chapter

- ♦ Graph node configuration
- ♦ About graph functions
- ♦ Test resolving graph nodes
- ♦ Transform label bindings and the cell parameter
- ♦ Transform graphs and nodes
- ♦ About transform graphs
- ♦ Types of Transform nodes

Transform Graphs provide a way to manipulate existing series data, like Niagara History records, in order to create graphs for reporting. The following topics describe some of the major concepts associated with the Series Transform Graph components.

Graph node configuration

You configure a graph node from its Property Sheet view, however, it is much easier to use the associated window editor for the graph node.

To open the appropriate window, double-click the component in the Wire Sheet view. These editors are the best way to configure the graph node, however, the Property Sheet view is still available for each graph.

For example, the following illustration adds scale factors for three separate history source value inputs. Enter a scale factor in the value field and click the **OK** button to save. Non-numeric fields (such as Status) are not available for scaling.

Figure 1 Scale Editor with three inputs

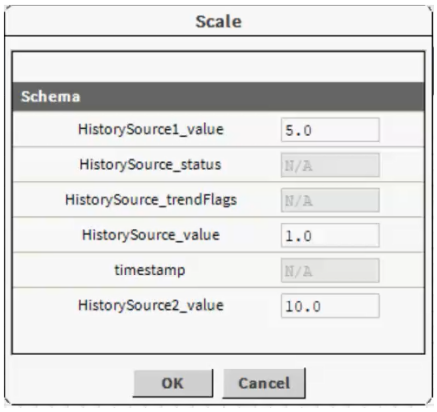
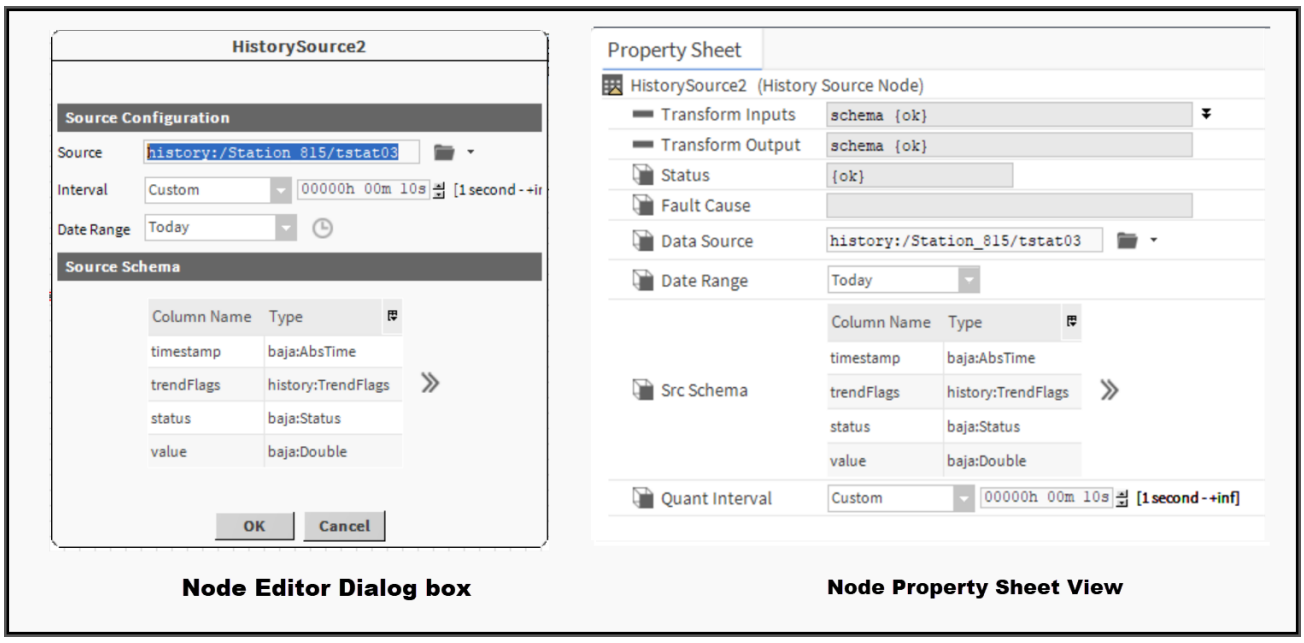


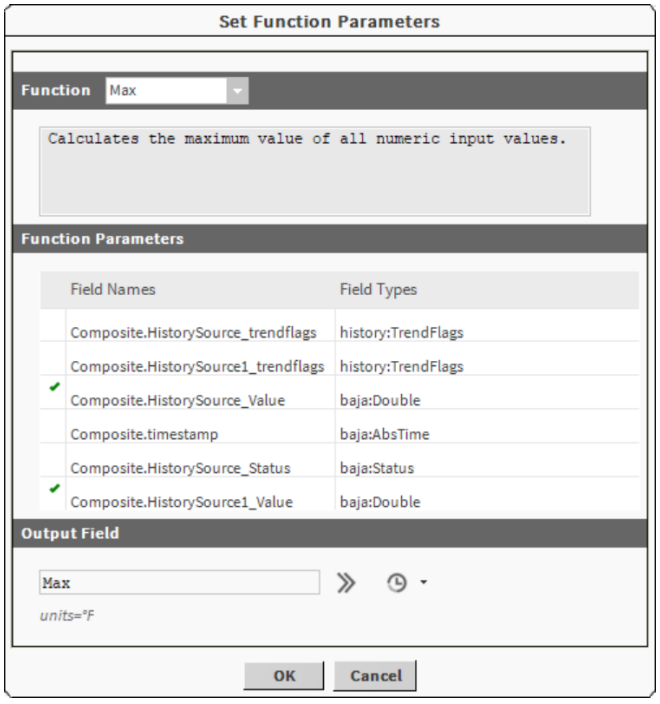
Figure 2 Node Editor window and property sheet views



About graph functions

Aggregate nodes and rollup nodes apply functions, such as Sum, Average, Max, and others, to the output. The function editor configures the functions assigned to the output schema field for either type of node.

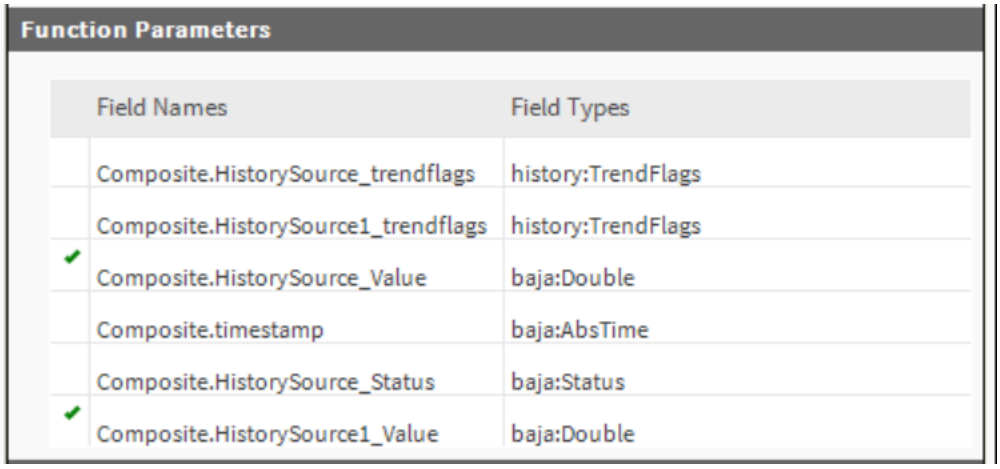
Figure 3 Function Editor



The function editor consists of three main sections:

- The function selector selects which function to assign to an output field. All available functions are contained in the function option list. Each function includes a function description.
- The parameter selector is a table of fields available as argument data sources. Each row of the table represents a schema field from an incoming source. Each row includes the "namespaced" schema field name, which includes the name of the source graph node component followed by the schema field name, and the schema field data type.

Figure 4 Function parameters



When the function cursor executes, the selected schema fields provide the argument data for the selected function.

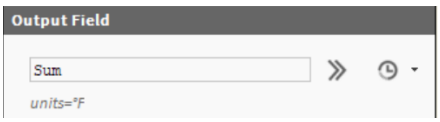
- When the Field Assignment editor executes, it assigns the results to the associated node schema output field. You type the field name into the output field text box editor. The function selection determines the field data type (the return type of the selected function).

  The assignment field editor includes a facets (BFacets) editor. In cases where the selected input fields of a function include different units (BUnit facet data) you need to select the unit type for the output field.

Example Facet assignment

As an example, assume you have two schema fields to use as data arguments for a Max function. The first field's values are units of Celsius, while the second field's values are units of Fahrenheit. To compare the two values, you must have a common unit type. You must also define the unit type of your result so that you can correctly interpret the data from the assigned function.

Figure 5 Output field

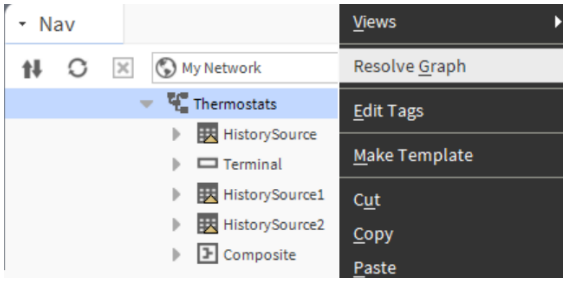


By assigning the unit type to the output field, you provide the common unit type used for comparison by the function at the time of cursor execution. You also determine the unit type used for the comparison result, which determines the actual value of the result data.

Test resolving graph nodes

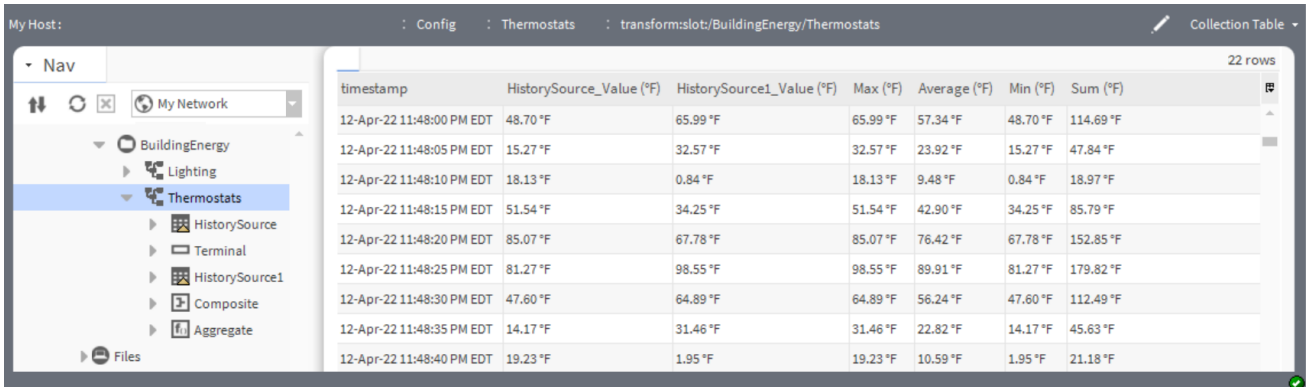
Each transform graph node includes a **Resolve Graph** menu option in the popup menu of the transform graph component. You can select this menu item to test your graph from Workbench so you know how it works before integrating it into a Px view. Select this action to resolve the graph against the data set as determined by the data source nodes of the graph (for example, the History Source node).

Figure 6 Resolve Graph menu item



The resolved data display in the Workbench table view. Use them to confirm the expected configuration results. If you get unexpected results, make changes and resolve the graph again.

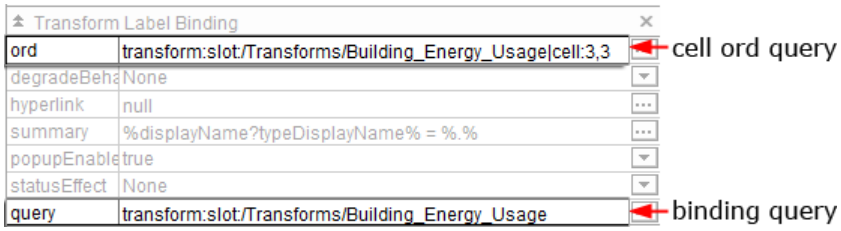
Figure 7 Transform graph resolved in collection table



Transform label bindings and the cell parameter

The transform label binding is similar to other bound label bindings with an additional query property and an ord query with which to designate a specific table cell.

Figure 8 Transform Label Binding



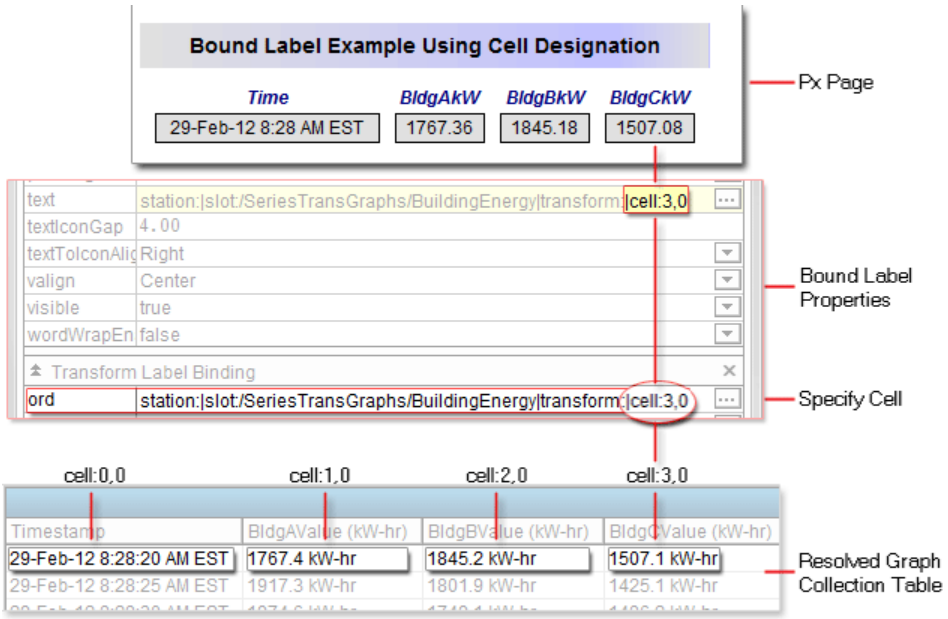
- The Binding query overrides select runtime properties of the graph nodes that make up the target transform graph. Overriding the values allows you to use the same graph in many Px files without having to change the transform graph itself.
- The Cell ord query specifies the column and row in the table that the selected transformgraph creates. If the designated binding does not produce tabular data, an error appears in Workbench and the ord is not resolved.

  Use the cell ord parameters (column value first and the row value second) to specify the column index and row index respectively. In the Make Widget Wizard you can use the Data Row and Data Column fields to add the parameters automatically. You can also manually edit an existing cell ord query, keeping the syntax: `|cell:2,0` at the end of the ord.

NOTE:

Both column and row are zero based indexes, for example 0,0 specifies the cell defined by the first column and first row in a table.

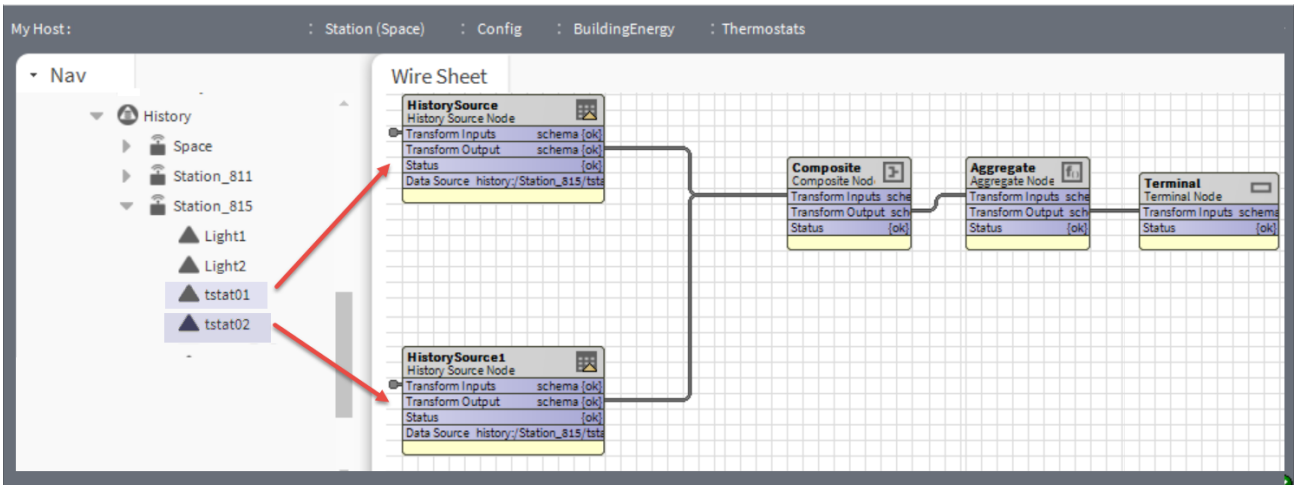
Figure 9 Bound label displays text from specified table cell



Transform graphs and nodes

Transform graphs comprise a collection of graph nodes contained as properties within the parent transform graph component. The default view of the transform graph component is the Wire Sheet view, where you work with the various nodes to create your report.

Figure 10 Series transform graph Wire Sheet view and source histories

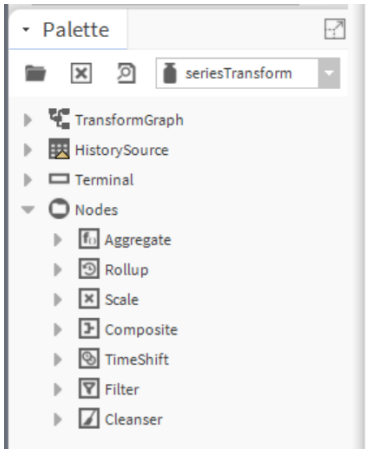


Typically, you assemble transform graphs by interconnecting graph nodes in the Wire Sheet view of the parent transform graph node, with source nodes and other nodes integrating into a final terminal node. On the Wire Sheet, each graph node connects into the graph by linking the output property of the component to the input property of the target component. Niagara histories provide the most likely source of tabular data. Each graph produces a single table result, as represented by a single terminal node.

As a third-party developer, you may create additional nodes using the series transform API. However, Niagara4 provides seven individual objects on the seriesTransform palette. These components provide four general categories of series transform objects:

- The Container (TransformGraph node) holds all the other graph nodes and has a **ResolveGraph** action that you use to generate the data transformation after you configure all the nodes.
- The Source (HistorySource node) is the single object that identifies the source data for your transformation.
- The Terminal (Terminal node) is the final node of any transform graph. Each transform graph must have one and only one terminal node.
- The Transforms (Transform nodes) provided on the palette include the Aggregate, Composite, Rollup, Scale, TimeShift, Filter and Cleanser nodes.

Figure 11 Series Transform Graph palette

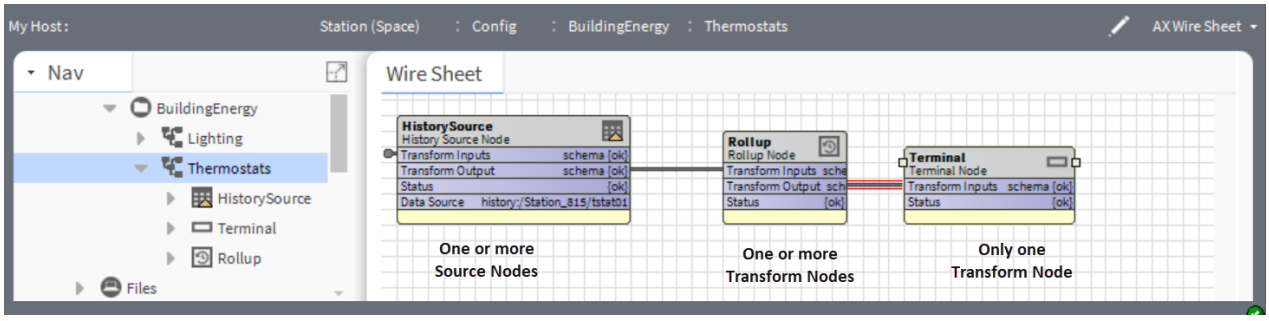


By adding components from the seriesTransform palette to a Transform Graph Wire Sheet view, you can create a static definition of how to transform run-time data into a new single-output set of information. You create the transform graph using HistorySource nodes, one or more transform nodes, and a single terminal node. By configuring the data schema of each transformation you design how the data flow from one transformation node to the next.

NOTE:

A transform graph container must contain all nodes and all nodes within the transform graph container must be uniquely named.

Figure 12 Minimum set of nodes on the seriesTransformGraph Wire Sheet view



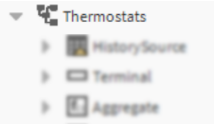
Each of the graph nodes described in the following sections includes a special popup editor that opens when you click the graph node component in the Wire Sheet view. To access the Property Sheet view, select it from the Workbench View Selector.

About transform graphs

A series transform graph is a node container that holds all other graph nodes. The transform graph node itself is also a graph node that contains other transform graphs. This makes it possible to create transform graphs that provide very specific data functionality, and use them as building blocks to create larger graphs.

After you add a transform graph node to your station, double-clicking on it opens the node's default Wire Sheet view where you can configure graph properties. When properly configured, a series transform graph performs a transformation or series of transformations against a set of data to produce a single result.

Figure 13 Transform graph component



Working with transform graph nodes:

- To define the data schema, each node must contain a single terminal node plus one or more source nodes. Source nodes may be HistorySource nodes or other transform graph nodes.
- Each node returns a single result and thus contains a single terminal node.
- Nodes may contain one or more transformation-type node(s), such as the Aggregate, Rollup, Scale, or Composite nodes.
- Transform graphs may be included as transformations within other transform graphs.
- To resolve a configured transform graph node in Workbench (to see a table view of the source data) right-click on the transform graph node in the Nav tree and select **Resolve Graph** from the popup menu.
- To see a configured transform graph node as a chart or as a table in a Px page, use the PxEditor Make Widget Wizard.

The following illustrations show some example transform graph implementations.

Figure 14 A graph containing graph nodes

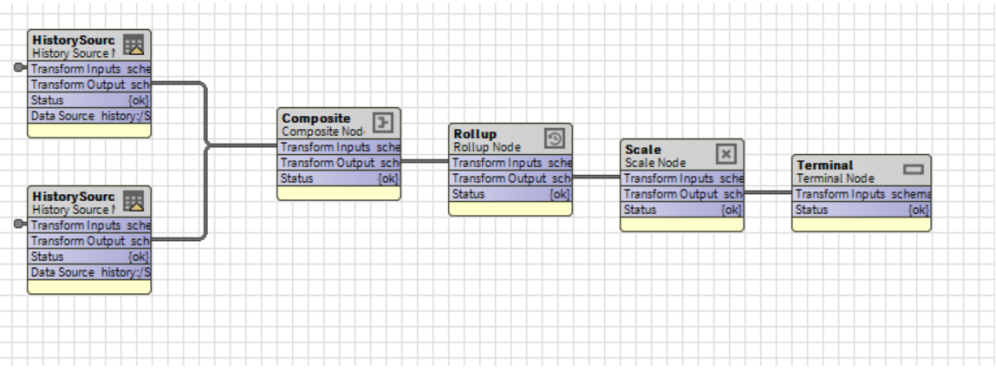


Figure 15 A graph containing another graph as a source

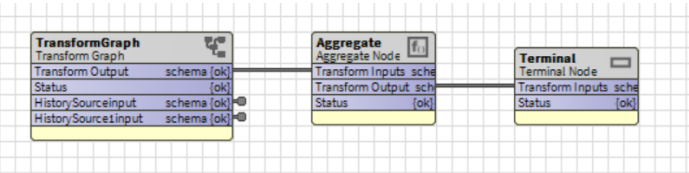
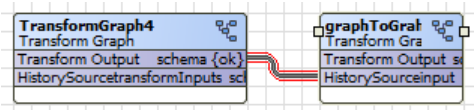


Figure 16 Two interconnected graphs. The first graph acts as data source for the second.

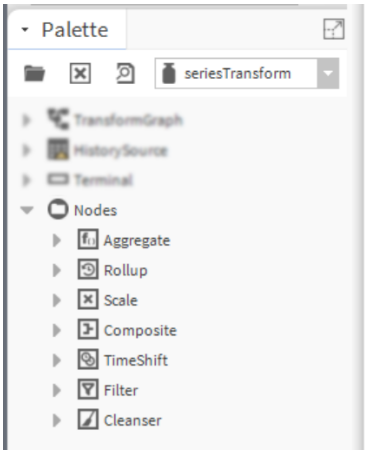


Types of Transform nodes

Transform nodes are located in the Nodes folder of the **seriesTransform** palette.

They include the following:

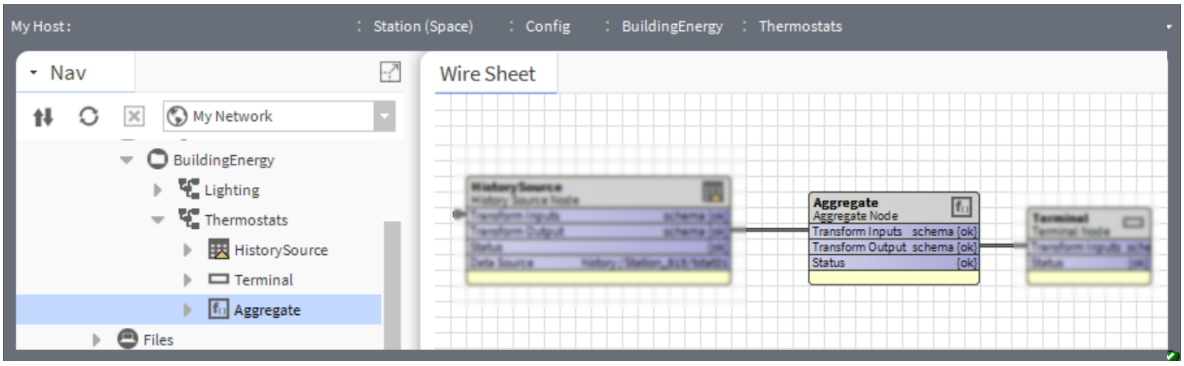
Figure 17 Transform nodes located in the seriesTransform palette



About the Aggregate node

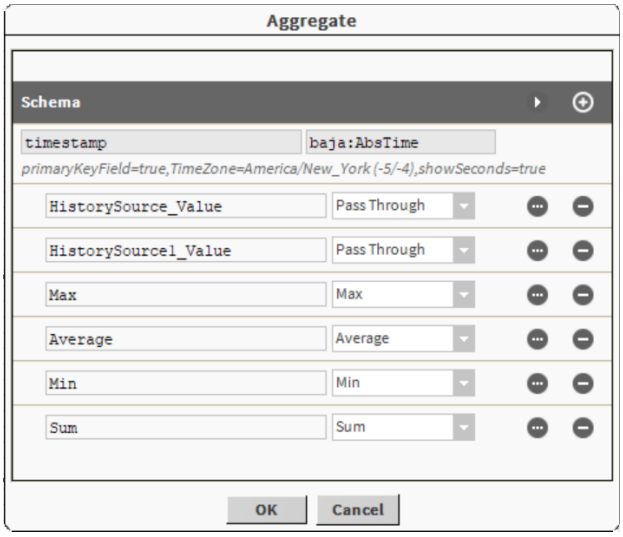
The Aggregate node performs functional operations against each row of data produced by the node data source. These functional operations do not take into account the order of the parameters, and so are limited to functions dealing with input values as an aggregate.

Figure 18 Aggregate node



The **Aggregate Editor**, like the **Composite Node** editor, creates the output schema of the aggregate node. To create the schema, you add field names to the schema and assign functions to each property.

Figure 19 Aggregate Editor

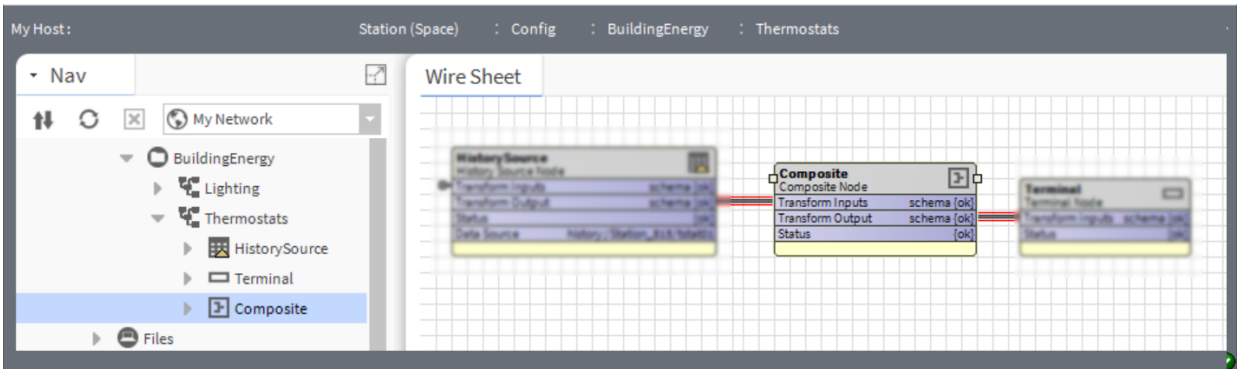


Each aggregate function takes a set of input source fields as parameter values. You select these input parameters in the **Function Editor**. There is no limit to the number of parameters that you can select for a function, with the exception of the Pass Through function, which passes through a single value without altering it. This makes the value available for the next consuming node in the graph.

About the Composite node

This node combines more than one input source into a single data structure. You can include fields of input sources multiple times in the composite schema so long as you name them uniquely each time. In addition to using multiple HistorySource nodes, you can include a single incoming schema element multiple times in the Composite node.

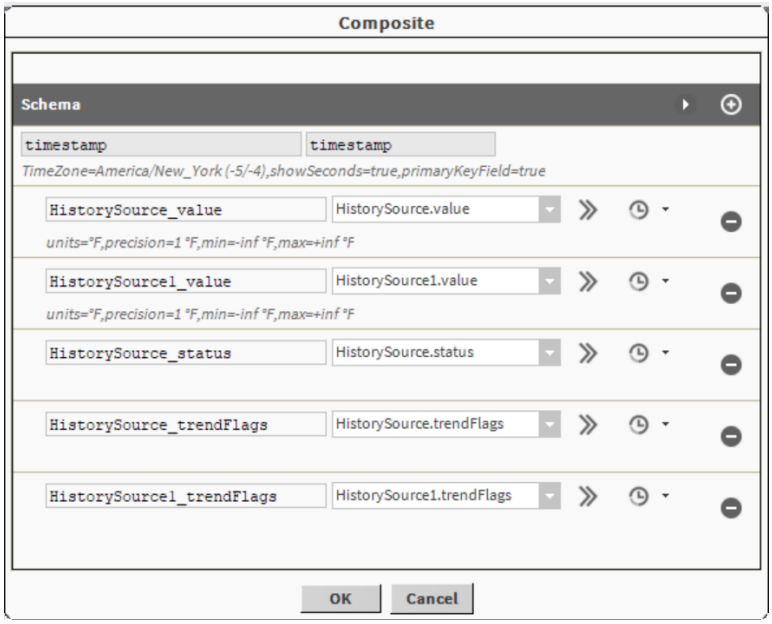
Figure 20 Composite node



Compositing data into larger structures allows for comparing data in tandem, as well as performing operations over multiple data sets to create new information.

An example of this would be creating a composite of two history sources and using the aggregate node to determine the Max value between the two value fields for each data row. You can use the composite editor to configure the composite schema. Do this by adding fields to the composite schema and then setting the values as fields from one of the source schemata.

Figure 21 Composite Editor



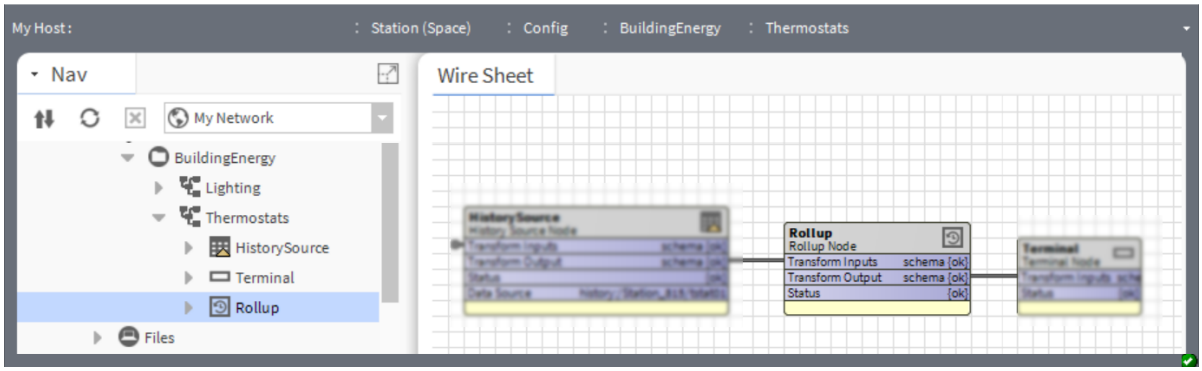
When you click the **Add** button (lay), an empty text field and option list open in the editor. To complete the addition of a field, first type a unique name in the text property for the schema field, then select an input field from the drop-down list. Each field in the option list is a field from one of the incoming schemata.

The **Composite Editor** includes an **Auto** button ( ) in the top right corner of the editor. Clicking this button populates the composite schema with fields from each incoming schemata, automatically supplying unique field names.

About the Rollup node

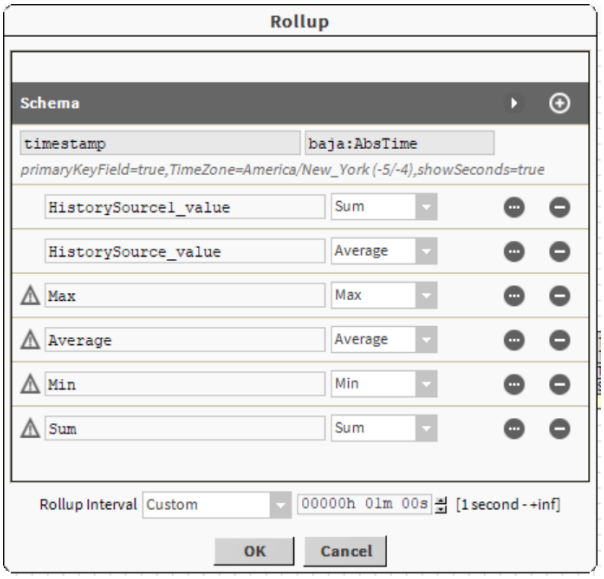
Like the Aggregate node, this node performs functional operations against each row of data produced by linked data sources. Unlike the Aggregate node, it can operate against several rows of data. The number of rows used to collect function argument data is determined by the rollup interval size.

Figure 22 Rollup node



You can use the **Rollup Editor** to create the output schema of the graph node. You create the schema by adding field names to the schema and assigning functions to each field.

Figure 23 Rollup Editor



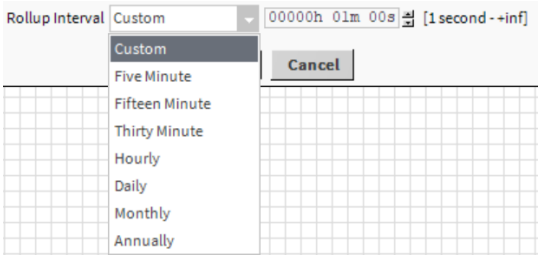
When you click the **Add** button ( ), an empty text field and drop-down list open in the editor. To complete the addition of a field, first type a unique name in the text field for the schema field and then select an input field from the drop down-list. Each field in list is a field from one of the incoming schemata.

The **Rollup Editor** includes an **Auto** button ( ) in the top right corner of the editor. Clicking this button populates the rollup schema with fields from each incoming schemata, automatically supplying unique field names.

Each function takes a set of input source schema fields as parameter values. You use the Function Editor to select parameters to apply. There is no limit to the number of parameters you may select for a function, with the exception of the Pass Through function, which passes through a single value without altering it. This makes the value available for the next consuming node in the graph.

The Rollup interval is an absolute time-based interval that assumes the key field of the incoming data schema. You can use the Rollup Interval option list to select a rollup interval. The interval selector provides several predefined settings and also provides a custom setting the includes a custom time editor.

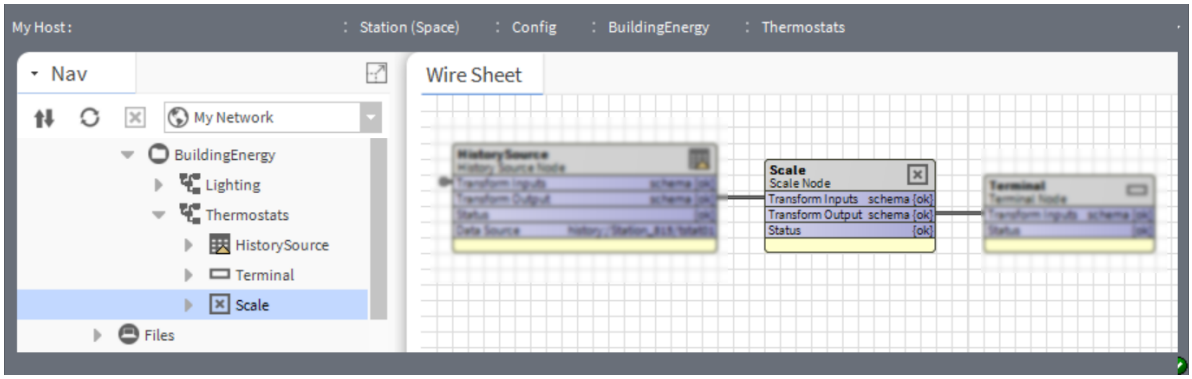
Figure 24 Rollup Interval selector



About the Scale node

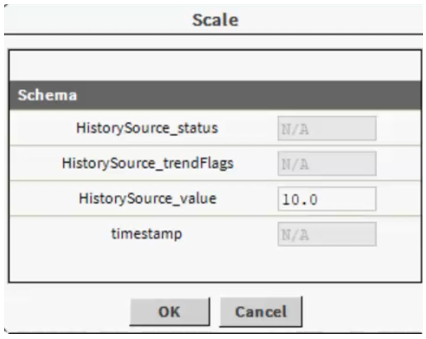
The Scale node selects numeric input fields from a HistorySource node to create an Output that is the product of the Input and the Scale factor. You cannot modify non-numeric data (such as Status). You can link the scaled output to the Input of a terminal.

Figure 25 Scale node



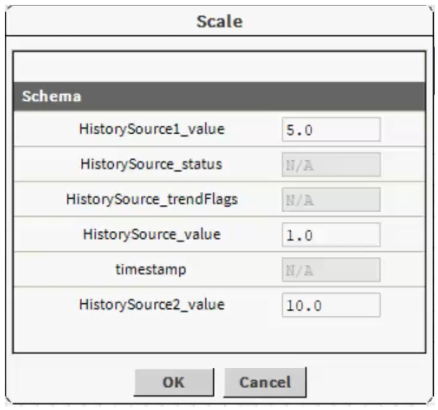
The Scale Editor lets you add a scale value in the value field for any numeric data.

Figure 26 Scale Editor



You can use the **Scale Editor** to assign scale factors for each input. For example, the following illustration adds scale factors for three separate history source value inputs. Enter a scale factor in the value field and click the **OK** button to save. Non-numeric fields (such as Status) are not available for scaling.

Figure 27 Scale Editor with three inputs



About the Time Shift node

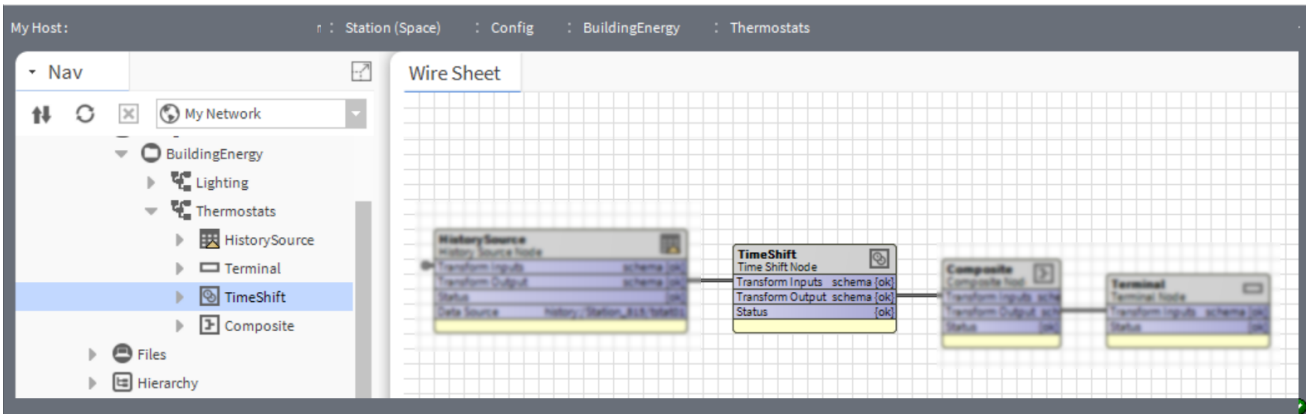
The TimeShift node adjusts the time values of timestamps on a set of data generated to a SeriesTransformTable (using the TransformGraph component).

To use the TimeShift node, drag it from the Series Transform palette onto the Wire Sheet view and connect a HistorySource output to the TimeShift Inputs slot. Double-click on the TimeShift component in the Wire Sheet view and set the desired time shift value in the **TimeShift** window.

Figure 28 Set the time shift value in the TimeShift window

Connect the output from the TimeShift node to another SeriesTransform input or directly to a Terminal Node input as desired.

Figure 29 TimeSheet node on Wire Sheet



Resolve the graph and check that the table displays time values that are offset by the value you set in your **TimeShift** window. So, for example, if you shifted your time by setting a 30-minute value in the TimeShift window, history records that start at 15:30 hrs in the actual point history table should be offset by 30 minutes (16:00 hrs) in the resolved graph table.

The TimeShift node works with the following data types:

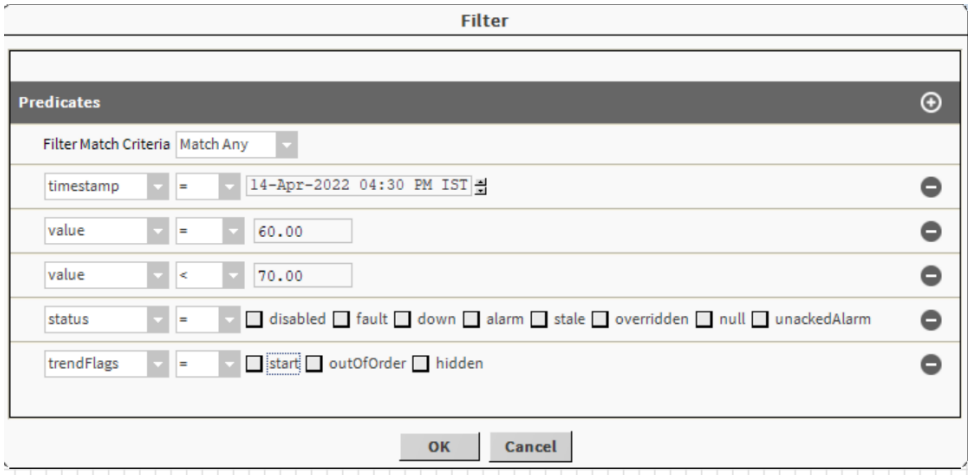
- Numeric Points
- Boolean Points
- Enum Points
- String Points
  - CoV Type histories
  - Interval type histories

About the Filter node

The Filter node (BqlFilter Node) filters a series of data using a bql predicate expression. For example, you can use an expression such as: `value > 80` to show only values greater than 80 or `timestamp in bqltime.yesterday` to limit values to those recorded on the previous day.

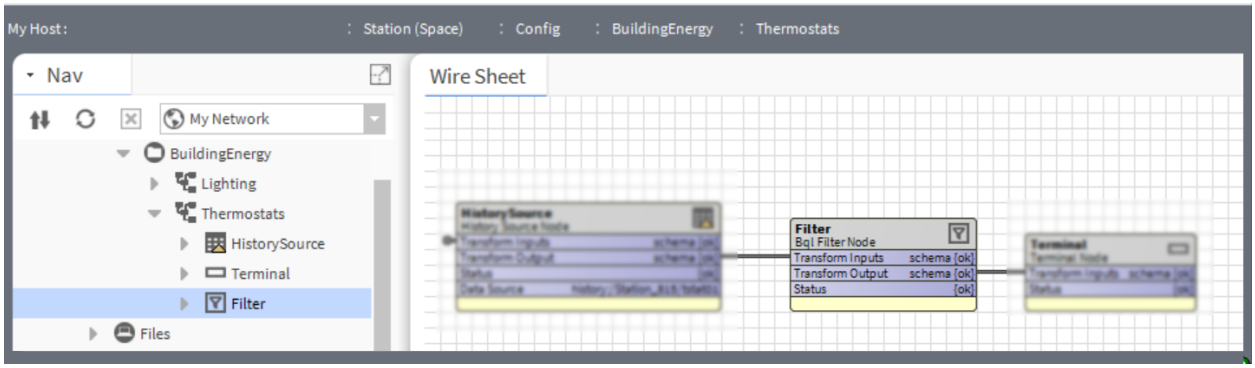
To use the FilterNode component, drag it from the Series Transform palette onto the Wire Sheet view and connect a HistorySource output to the Filter Inputs slot. Double-click on the Filter component in the Wire Sheet view and use the **Filter** window to create a custom bql query.

Figure 30 Set the Filter values in the Filters window



Connect the output from the Filter node to another SeriesTransform node input or directly to a Terminal Node input as desired.

Figure 31 Filter node on the Wire Sheet view



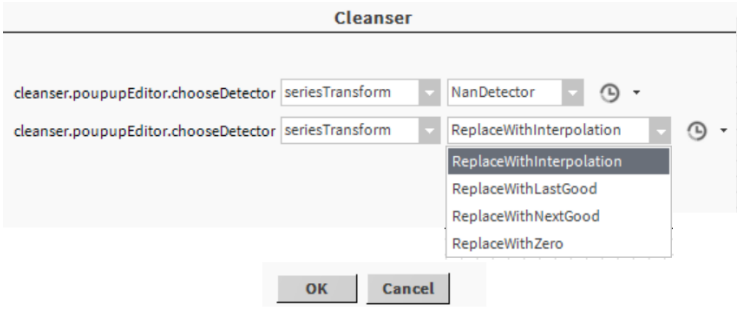
Resolve the graph and check that the table displays the expected time values as defined by the query set in the **Filter** window.

About the Cleanser node

The cleanser node provides a schema to replace the NaN values in the set of data source. It supports only for BINumeric value types which primarily represent a numeric value stored as a double. For example, if you set the fan switch to true or false, the cleanser node provides an error that says it "**works only with BINumerics stored in the "value" property**".

To use the CleanserNode component, drag it from the Series Transform palette onto the Wire Sheet view and connect a HistorySource output to the Cleanser Inputs slot. Double-click on the Cleanser component in the Wire Sheet view, and use the detectors and replacers in the **Cleanser** window to replace the NaN values in the data source.

Figure 32 Cleanser Editor

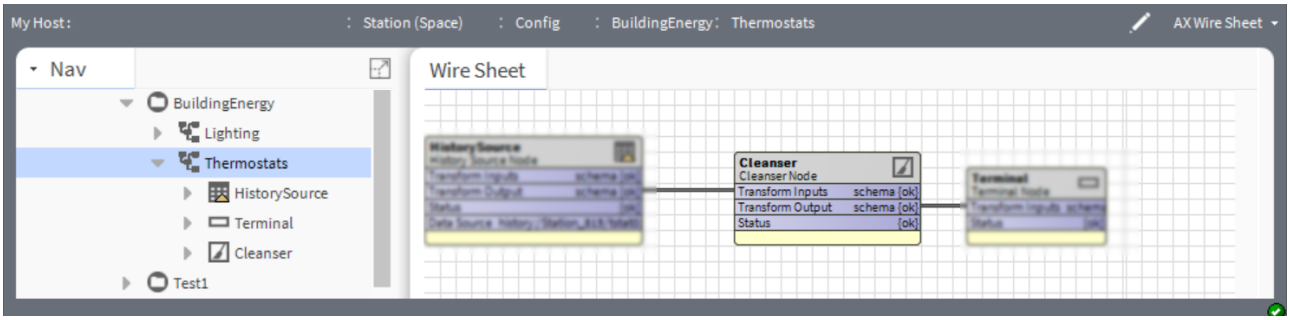


Connect the output from the Filter node to another SeriesTransform node input or directly to a Terminal Node input as desired.

The cleanser node have only one **NanDetector** and four replacers types.

- **ReplaceWithInterpolation**: If a NaN value is detected in the source data, replace with the average of the previous record and the next record which have valid non-NaN values
- **ReplaceWithLastGood**: If a NaN value is detected in the source data, replace with the previous record which has a valid non-NaN value.
- **ReplaceWithNextGood**: If a NaN value is detected in the source data, replace with the next record which has a valid non-NaN value.
- **ReplaceWithZero**: If a NaN value is detected in the source data, replace the NaN values with a zero value.

Figure 33 Cleanser Node on Wire Sheet



Resolve the graph by using the **NaNDetector** and different replacers in the cleanser editor and check the table it displays the numeric values without NaN values in the data source.

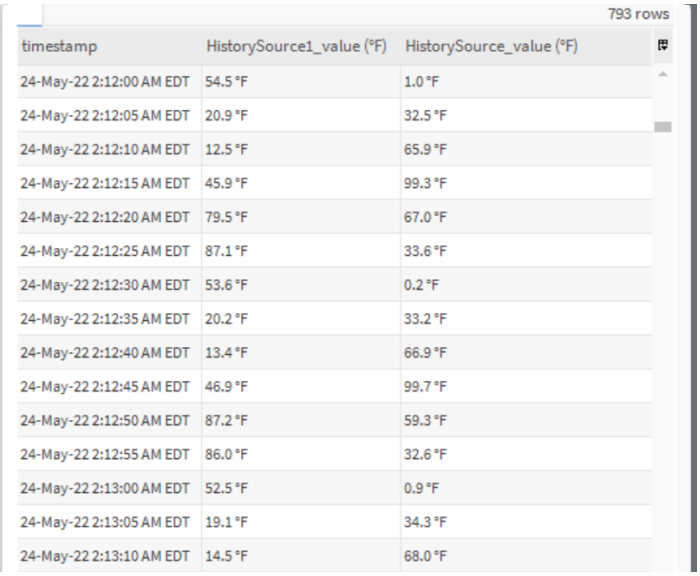
Chapter 2 Series Transform Graphs and Px views

Topics covered in this chapter

- Creating a series transform collection table in a Px view
- Creating a series transform bound table in a Px view
- Creating a series transform chart in Px view
- Creating a series transform bound label in a Px view

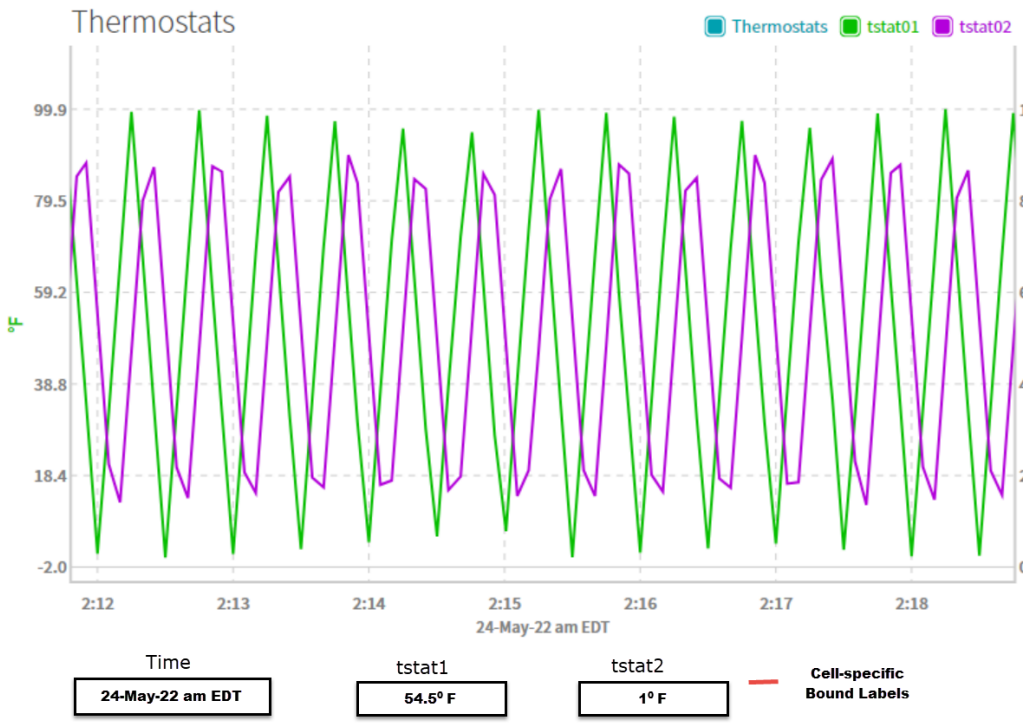
You can create Px page views with data from the seriesTransform component.

Figure 34 Series transform graph data in Px view (Collection Table, Chart, Bound Labels)



793 rows

timestamp	HistorySource1_value (°F)	HistorySource_value (°F)
24-May-22 2:12:00 AM EDT	54.5 °F	1.0 °F
24-May-22 2:12:05 AM EDT	20.9 °F	32.5 °F
24-May-22 2:12:10 AM EDT	12.5 °F	65.9 °F
24-May-22 2:12:15 AM EDT	45.9 °F	99.3 °F
24-May-22 2:12:20 AM EDT	79.5 °F	67.0 °F
24-May-22 2:12:25 AM EDT	87.1 °F	33.6 °F
24-May-22 2:12:30 AM EDT	53.6 °F	0.2 °F
24-May-22 2:12:35 AM EDT	20.2 °F	33.2 °F
24-May-22 2:12:40 AM EDT	13.4 °F	66.9 °F
24-May-22 2:12:45 AM EDT	46.9 °F	99.7 °F
24-May-22 2:12:50 AM EDT	87.2 °F	59.3 °F
24-May-22 2:12:55 AM EDT	86.0 °F	32.6 °F
24-May-22 2:13:00 AM EDT	52.5 °F	0.9 °F
24-May-22 2:13:05 AM EDT	19.1 °F	34.3 °F
24-May-22 2:13:10 AM EDT	14.5 °F	68.0 °F



The above screen capture shows a Px page that uses different widgets to display the resolved series transform graph.

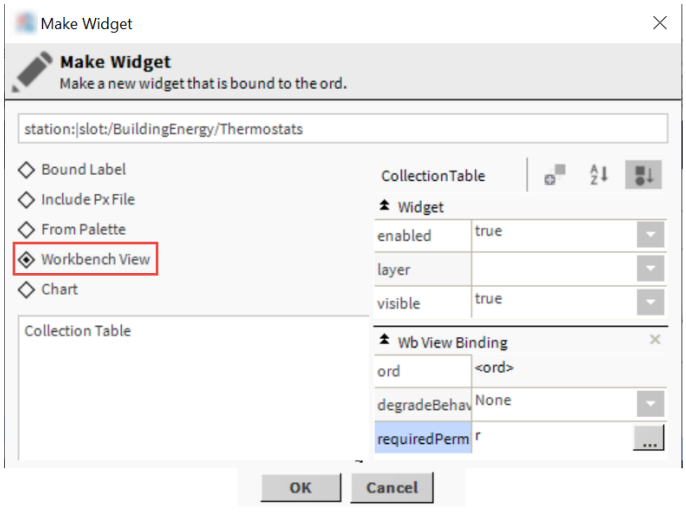
Creating a series transform collection table in a Px view

You can add a series transform collection table view to a Px view using the Make Widget wizard.

Step 1 Drag the SeriesTransformGraph component from the nav tree onto a Px page.

The Make Widget Wizard opens.

Figure 35 Choosing a Collection Table to add to a Px view



Step 2 Select the `Workbench View` option in the Make Widget Wizard, choose `Collection Table`, and click the **OK** button.

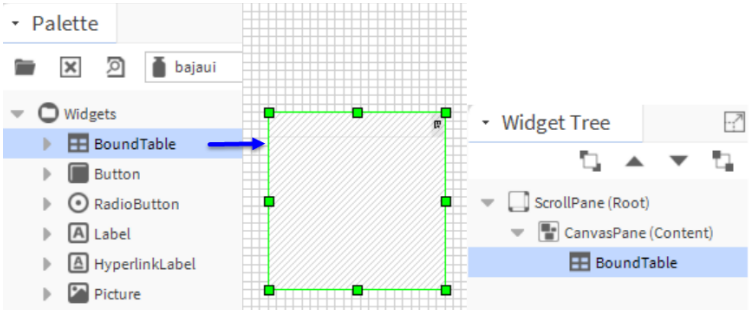
The Collection Table view displays in the Px view.

Creating a series transform bound table in a Px view

You can add a series transform bound table to a Px view using the BoundTable widget.

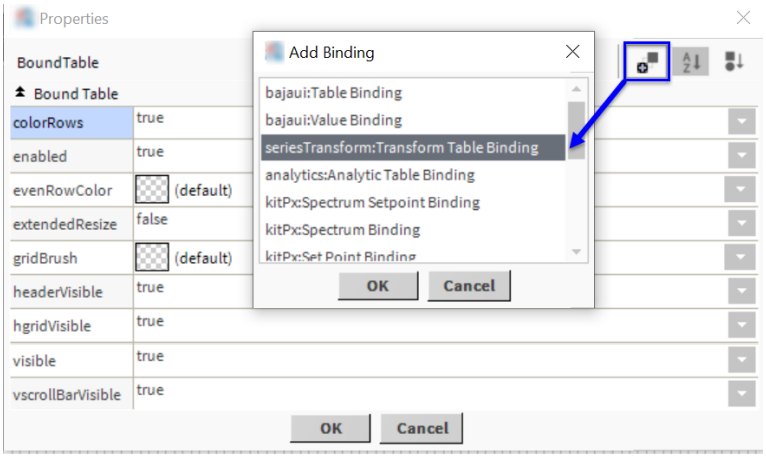
Step 1 Drag a BoundTable widget from the bajui palette onto your Px page in the Px Editor view.

Figure 36 Add BoundTable from bajaui palette



Step 2 Double-click on the BoundTable component in the WidgetTree palette to open the **Properties** window.

Figure 37 Add Transform Table Binding



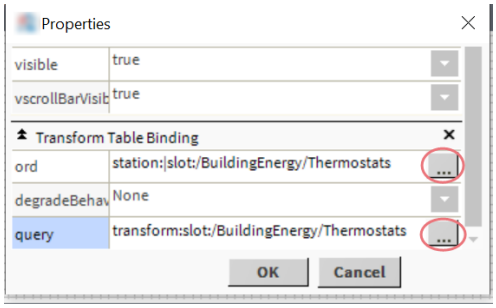
Step 3 Click the **Add Binding** button on the **Properties** window, choose `seriesTransform:Transform Table Binding` from the **Add Binding** window and click **OK** in the **Add Binding** window only.

This adds the Transform Table Binding to the bottom of the **Properties** window.

Step 4 In the Transform Table Binding rows of the **Properties** window, click the edit button on the ord row.

The **ord** window opens.

Figure 38 Add Transform Table Binding



Step 5 In the **ord** window, use the Component Chooser option to browse to the desired Transform Graph component, select the component and click the **OK** button.

The ord binding is set.

Step 6 In the Transform Table Binding rows of the **Properties** window, click the **Edit** button on the query row.

The **Select Ord** window opens.

Step 7 In the **Select Ord** window, use the Component Chooser option to browse to the (same) desired Transform Graph component, select the component and click the **OK** button.

The query binding is set and the Graph Parameters fields display with editable fields.

Step 8 Close all windows to complete the setup and display the Bound Table on Px page.

Creating a series transform chart in Px view

You can create a chart view of a resolved seriesTransform graph.

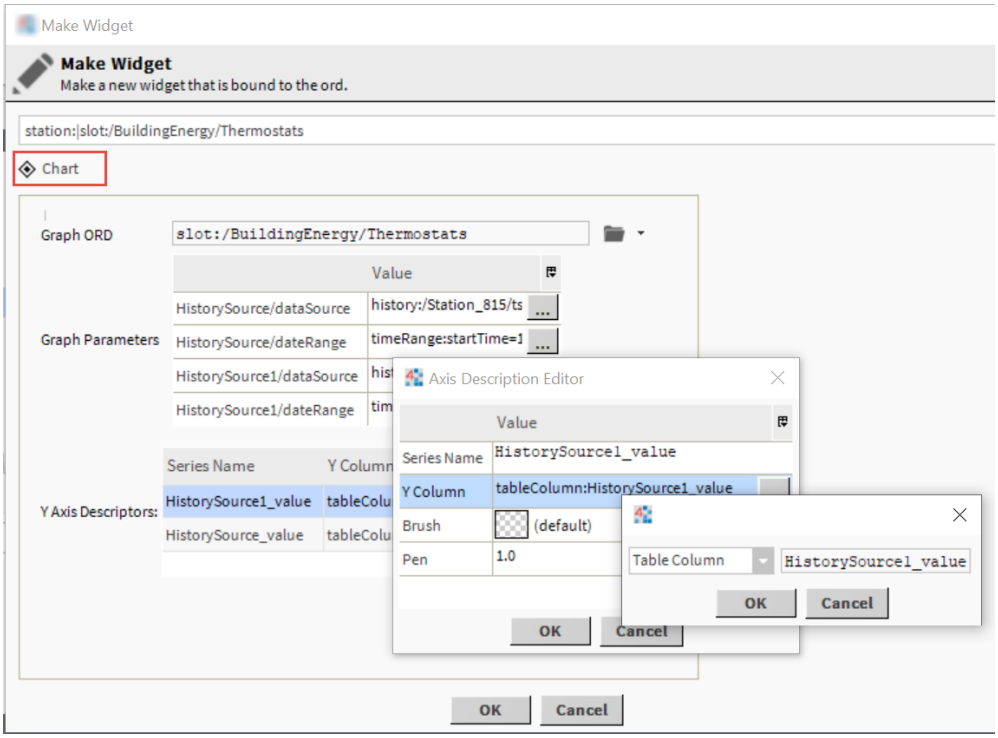
Step 1 Drag the SeriesTransformGraph component from the nav tree onto a Px page in the Px Editor.

The Make Widget Wizard opens.

Step 2 Select the `Chart` option in the Make Widget Wizard and verify that the Graph ORD field and Graph Parameters are correct.

Step 3 In the Y Axis Descriptors field, add, delete, or edit Y columns as desired using the edit control buttons to the right of the field editor. You can only add columns for those values that are already a part of the output schema.

Figure 39 Editing Y Column for Series Transform Chart in Make Widget Wizard



Editing or adding a descriptor requires that you include:

- Series Name

  This is the name that is used in the legend of the chart for the axis.

- Y Column Name

  This is the axis name and is the name of the target graph's output schema field.

  NOTE:

  The Column Name must match the output parameter exactly, including the text case.

Step 4 After adding all the Y columns, click the **OK** buttons to close all the windows and complete the chart.

Creating a series transform bound label in a Px view

You can create a bound label view of a resolved series transform graph using the Make Widget Wizard.

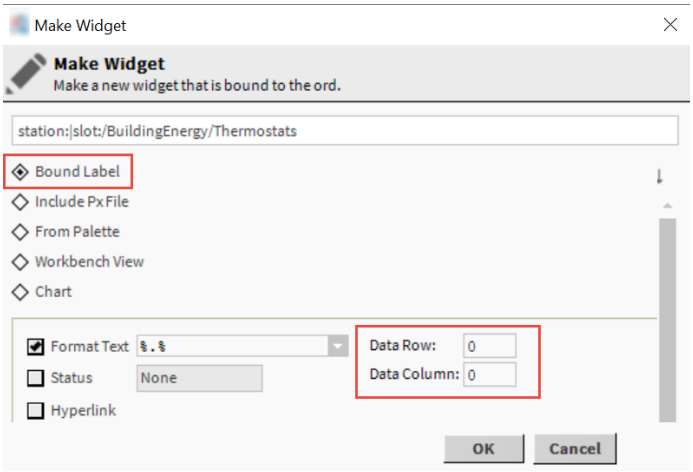
Step 1 Drag the SeriesTransformGraph component from the nav tree onto a Px page in the Px Editor.

The Make Widget Wizard opens.

Step 2 Select the `Bound Label` option in the Make Widget Wizard and verify that the Graph ORD field and Graph Parameters are correct.

Step 3 In the Data Row and Data Column fields enter the row and column numbers that identify the table cell that you want to display and click the **OK** button.

Figure 40 Adding the row and column to identify the table cell



The bound label displays on the Px page.

Chapter 3 Components

Topics covered in this chapter

- seriesTransform-TransformGraph
- seriesTransform-HistorySourceNode
- seriesTransform-SeriesSchema
- seriesTransform-SeriesInterval
- seriesTransform-TerminalNode
- seriesTransform-TerminalMap
- seriesTransform-AggregateNode
- seriesTransform-RollupNode
- seriesTransform-FunctionMap
- seriesTransform-RollupInterval
- seriesTransform-ScaleNode
- seriesTransform-ScaleFactors
- seriesTransform-CompositeNode
- seriesTransform-Filter(BqlFilter)Node
- seriesTransform-CompositeMap
- seriesTransform-TimeShiftNode
- seriesTransform-CleanserNode

Components include services, folders and other model building blocks associated with a module. You may drag them to a property or wire sheet from a palette.

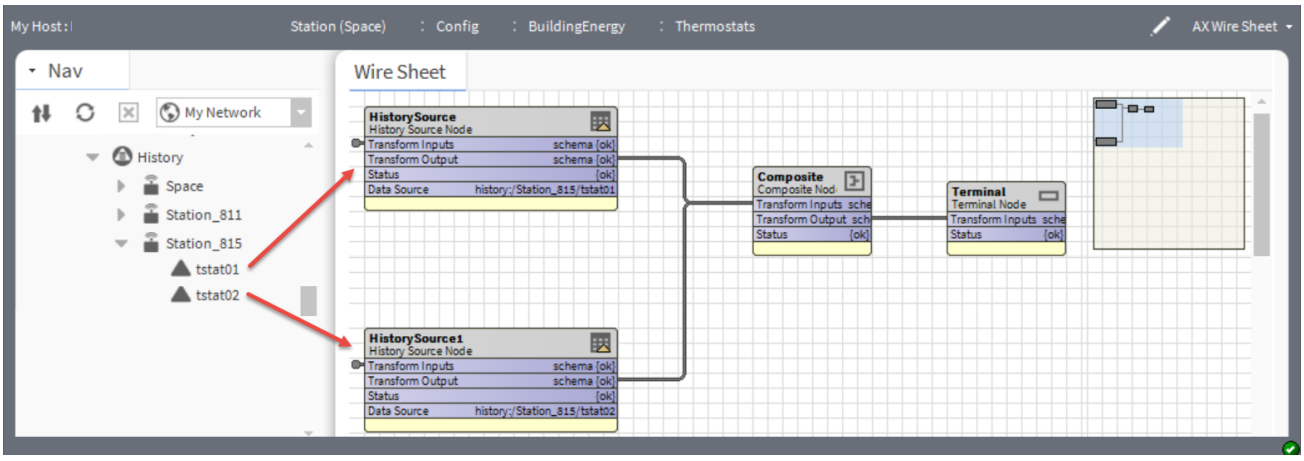
Descriptions included in the following topics appear as context-sensitive help topics when accessed by:

- Right-clicking on the object and selecting **Views→Guide Help**
- Clicking **Help→Guide On Target**

seriesTransform-TransformGraph

This object is a Wire Sheet that holds all the other graph nodes and has a **ResolveGraph** action that you use to generate the data transformation after you have configured all the nodes.

Figure 41 Series transform graph Wire Sheet view and source histories

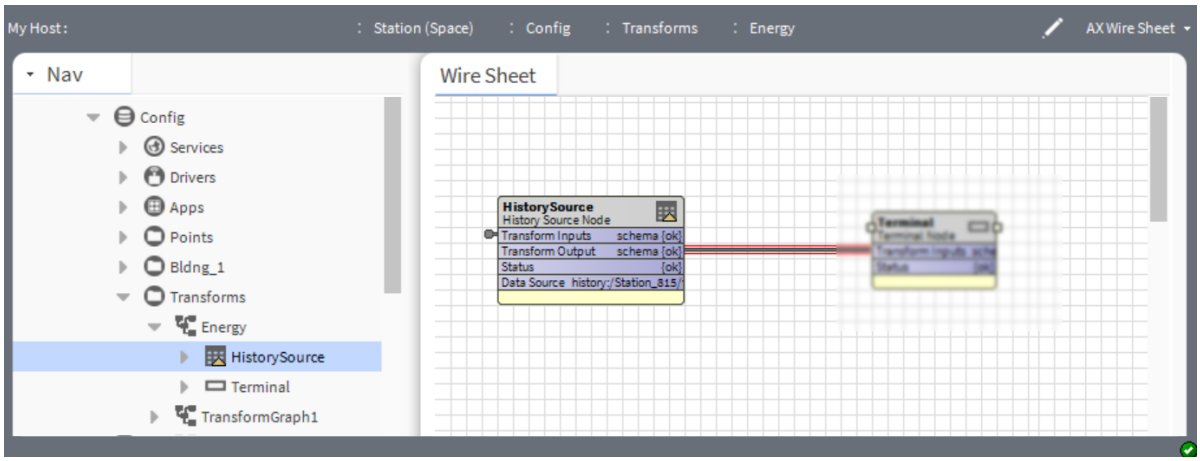


Like other components, you connect (or wire) these graph nodes together using the Wire Sheet view. Unlike other components, the wire connections indicate a flow of data from one node to the next rather than a means of setting property values.

seriesTransform-HistorySourceNode

This node is the single object available for identifying the source data for your transformation. It defines the data schema for the information to be processed by the transform graph (most often a Niagara history).

Figure 42 HistorySource node



Unlike other graph nodes, this node does not have an input property displayed in the Wire Sheet. This is because the node itself is a data source and will use the node configuration to obtain the source data. Place the HistorySource node on a transform graph node.

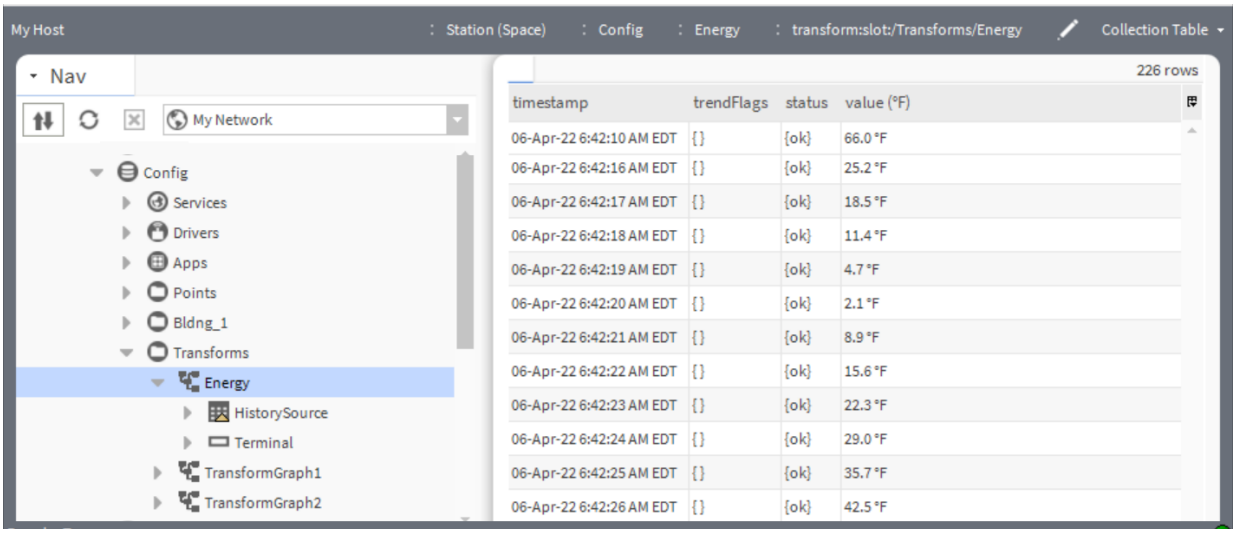
Each graph may have more than one HistorySource node with each node defining a single input for the graph. You may provide the actual source at graph resolution time as a graph property. You must set the value of this property when the graph resolves. The property value can be set manually, by a program object, or by an interface, such as the transform chart binding.

seriesTransform-SeriesSchema

This component defines the structure of some piece of data.

Fundamental to the series transform graph is the concept of a data schema that defines this structure. The several columns of Niagara history data displayed by a table view provide a common example of a data schema. In the following table, the columns: timestamp, Trend Flags, Status, and Value together form the history schema.

Figure 43 History table columns represent the history schema



Each history column has an associated name and data type. The **Timestamp** column is always an absolute time (`BAbsTime`) value. The data type of the **Value** column is based on the type of data the history represents. If the history serves a Boolean point, the **Value** column contains Boolean data. If the point is a numeric point, the value contains numeric data.

You configure each graph node on the Wire Sheet to create an output schema for other graph nodes to use when setting up a data transformation. This is what makes the process of manipulating history data and other forms of data possible. To manipulate data, such as the data in a history, you must know the point's data type (format). This allows you to assign different value transformations and calculations to specific data.

As an example, imagine that you want to scale the value of a history record by a factor of 3.9. How would you know that the **Value** column can be scaled? If the **Value** column contains Boolean data, you cannot scale the data by number (what is 3.9 times False?). By knowing both the name and the data type of the column to manipulate, you can create intelligent data transformations that give information about that data.

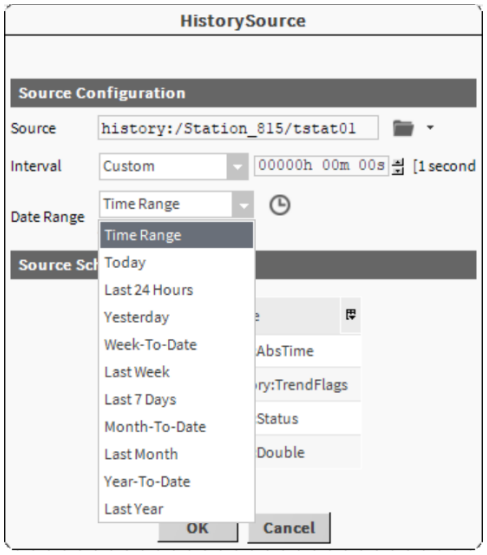
When designing the graph node, you only need the output schema of the source graph node to configure the consuming graph node. By configuring your graph node against the input graph node schema, you are configuring how the graph node processes data returned by the source graph node at the time the station executes the graph.

seriesTransform-SeriesInterval

When creating a transform graph, it is important that the data conform to exact intervals to allow data from different history sources to properly compare and combine into larger data sets. To do this, a process of "quantization" is used to fit the timestamp value of a record to an exact interval.

In the case of histories, the timestamp of each history includes not only the hour, minute, and second that the history was created, but also the millisecond value. When attempting to align different histories for comparison, this difference in milliseconds prevents an exact comparison. The **History Source** window can set intervals to predefined increments using the **Interval** option list.

Figure 44 Interval and Date Range fields



Setting the Interval to `Automatic` selects the current interval for history collection. For example, if the history is already set to collect at 15 minute intervals, the framework also sets the `Automatic` quantization interval of the history source node to 15 minutes.

NOTE:

All values are quantized based on a starting time of `0:0:0.000`. This means that an interval of 15 minutes quantizes a record whose timestamp reflects the value of `11:13:52.093` to `11:00:00.000`.

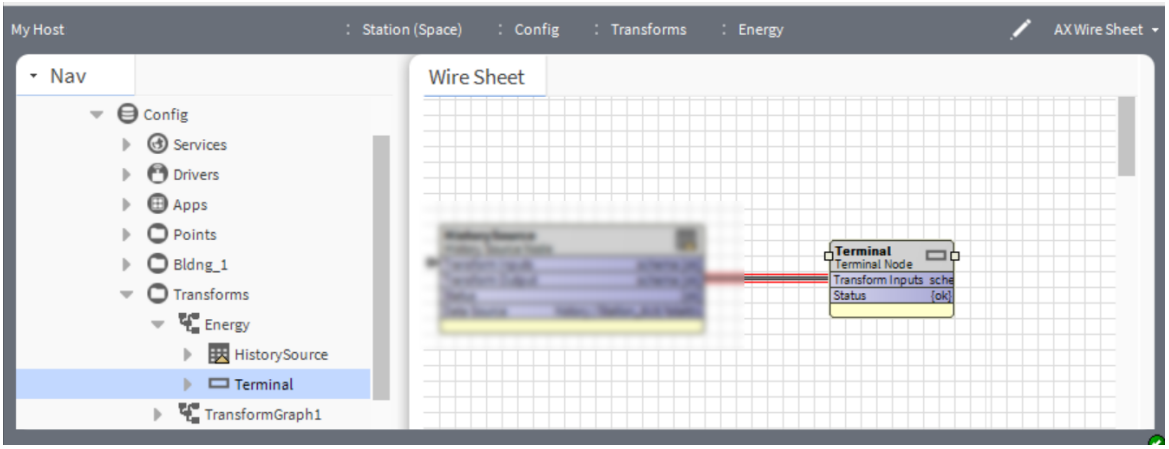
To use only a segment of a History rather than a point value's entire history, use the Date Range property and select the starting and ending date and times for the history data. The framework includes in the data set at the time the graph is resolved all history records between these date values, including those histories that fall exactly on those date values.

seriesTransform-TerminalNode

This component is the final node of any transform graph. Each transform graph must have one and only one terminal node.

The Terminal node designates the result schema of a transform graph. Each graph must include one and only one terminal node. The Terminal editor configures the order of the resulting graph's schema fields (data columns). This is especially useful when testing the graph result using the **Resolve Graph** Nav menu option of a transform graph.

Figure 45 Terminal node



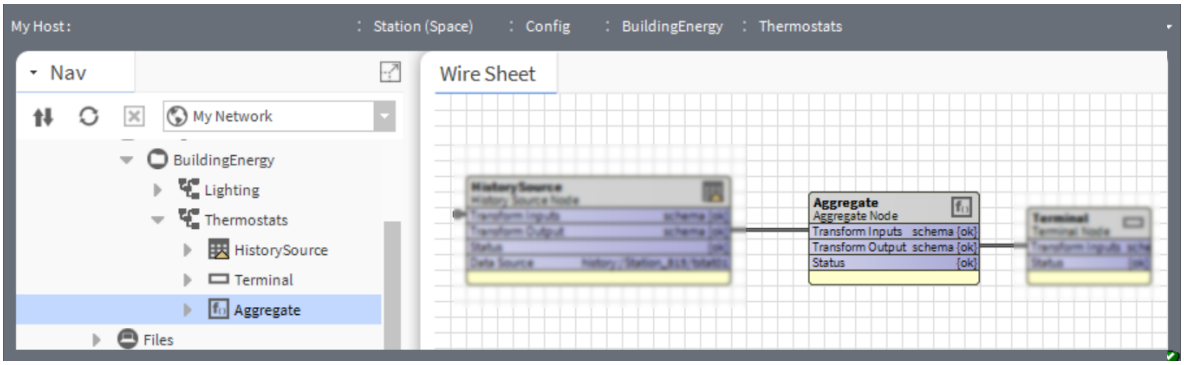
seriesTransform-TerminalMap

The TerminalMap component represents the mapping of source node values to the terminal. You use the Terminal Editor to arrange the order of these properties.

seriesTransform-AggregateNode

The Aggregate node performs functional operations against each row of data produced by the node data source. These functional operations do not take into account the order of the parameters, and so are limited to functions dealing with input values as an aggregate.

Figure 46 Aggregate node

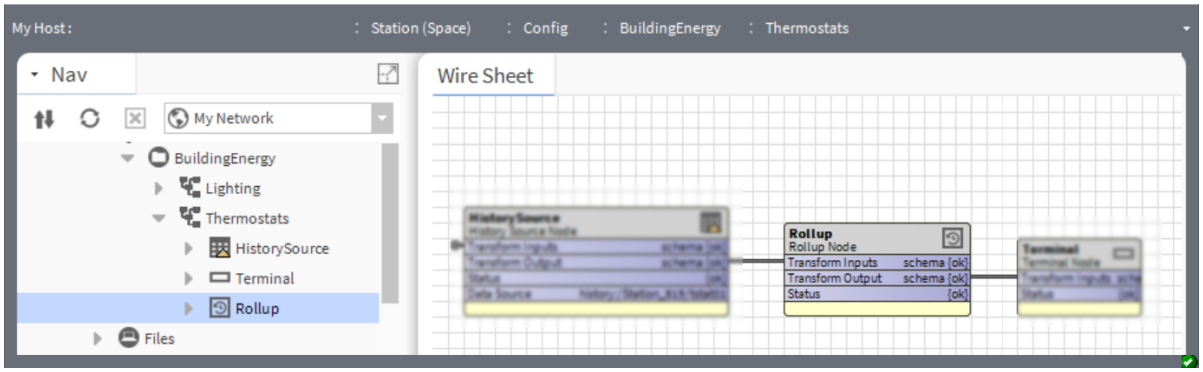


The **Aggregate Editor**, like the **Composite Node** editor, creates the output schema of the aggregate node.

seriesTransform-RollupNode

The Rollup node can perform functional operations against each row of data produced by linked data sources (similar to the Aggregate node). Unlike the Aggregate node, the Rollup node performs operations against several rows of data.

Figure 47 Rollup node



You use the **Rollup Editor** to create the output schema of the graph node. You create the schema by adding field names to the schema and assigning functions to each field.

seriesTransform-FunctionMap

The Function Map is associated with the Rollup and the Aggregate nodes. This component represents the mapping of source values against selected functions.

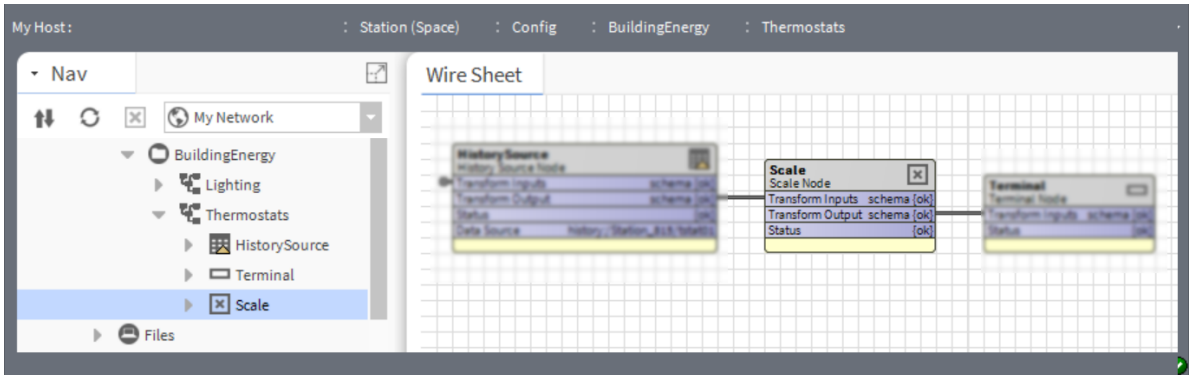
seriesTransform-RollupInterval

The Rollup Interval component holds the exact (custom or default option) time intervals that you select for a a particular Rollup node.

seriesTransform-ScaleNode

The Scale node multiplies selected numeric input fields from a HistorySource node to create an Output that is the product of the Input and the Scale factor.

Figure 48 Scale node



The Scale Editor adds the scale percentage as its value field for any numeric data.

seriesTransform-ScaleFactors

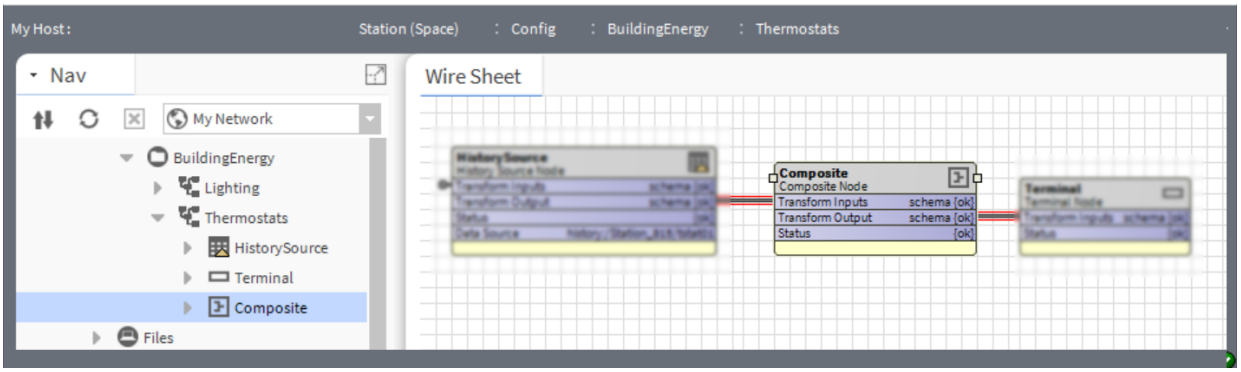
The Scale Factor component holds a number (one scale factor for one input) that you choose to multiply against a selected input value.

You cannot modify non-numeric data (such as Status). You can link the scaled output to the Input of a terminal.

seriesTransform-CompositeNode

This node combines more than one input source into a single data structure. You can include fields of input sources multiple times into the composite schema so long as they are uniquely named each time.

Figure 49 Composite node

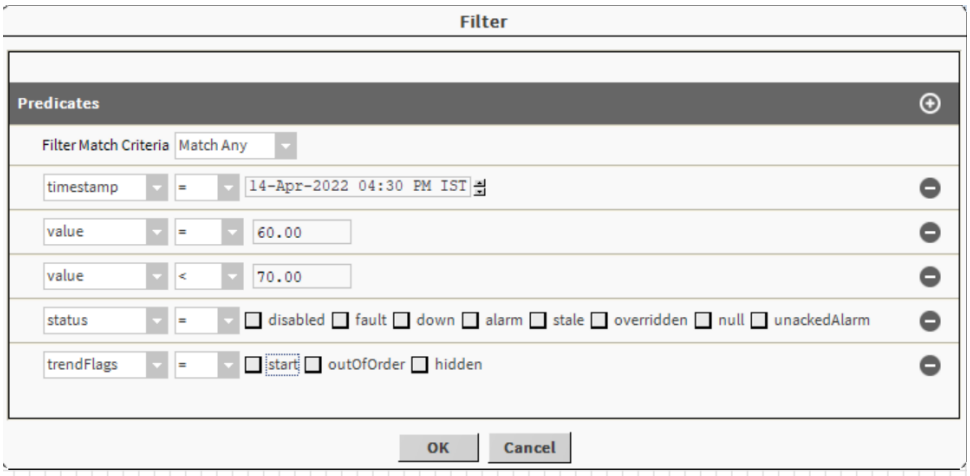


Compositing data into larger structures allows for comparing data in tandem, as well as performing operations over multiple data sets to create new information.

seriesTransform-Filter(BqlFilter)Node

The Transform Filter node provides a way for you to use a predicate query to refine your series transform graph data output. This component is available on the Series Transform palette.

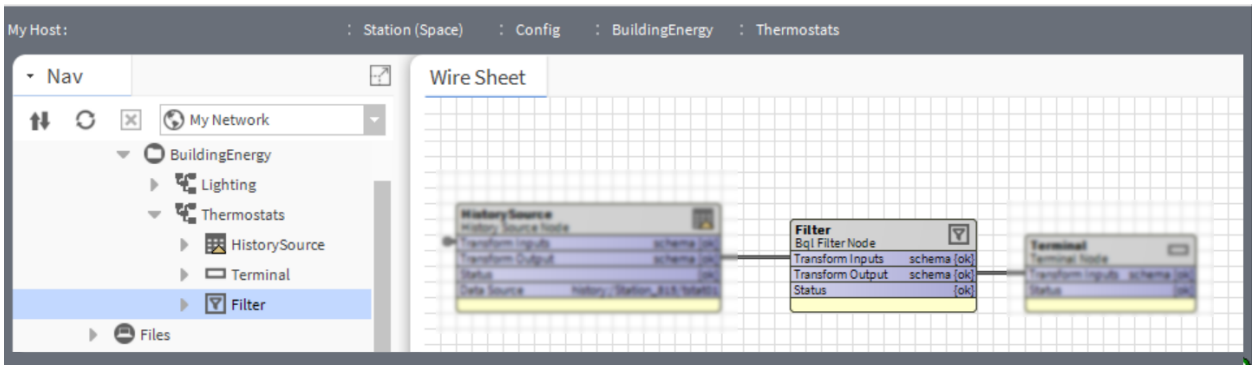
Figure 50 Set the Filter values in the Filters window



To use the FilterNode component, drag it from the Series Transform palette onto the Wire Sheet view and connect a HistorySource output to the Filter Inputs slot. Double-click on the Filter component in the Wire Sheet view and use the **Filter** window to create a custom bql query.

Connect the output from the Filter node to another SeriesTransform node input or directly to a Terminal Node input as desired.

Figure 51 Filter node on the Wire Sheet view



Resolve the graph and check that the table displays the expected time values as defined by the query set in the **Filter** window.

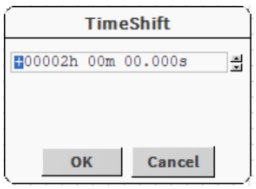
seriesTransform-CompositeMap

The Composite Map is associated with the Composite node. This component represents the mapping of the source node values to the composite node.

seriesTransform-TimeShiftNode

The Transform node adjusts the timestamp values that are resolved into a series transform table. The Time Shift node component is available on the Series Transform palette.

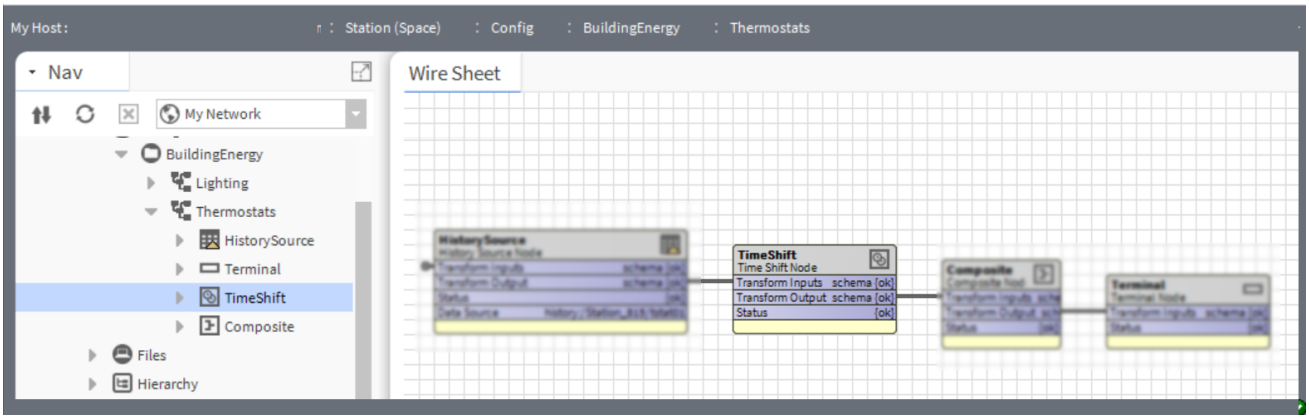
Figure 52 Set the time shift value in the TimeShift window



To use the TimeShift node, drag it from the Series Transform palette onto the Wire Sheet view and connect a HistorySource output to the TimeShift Inputs slot. Double-click on the TimeShift component in the Wire Sheet view and set the desired time shift value in the **TimeShift** window.

Connect the output from the TimeShift node to another SeriesTransform input or directly to a Terminal Node input as desired.

Figure 53 TimeSheet node on Wire Sheet



Resolve the graph and check that the table displays time values that are offset by the value you set in your **TimeShift** window. So, for example, if you shifted your time by setting a 30-minute value in the TimeShift window, history records that start at 15:30 hrs in the actual point history table should be offset by 30 minutes (16:00 hrs) in the resolved graph table.

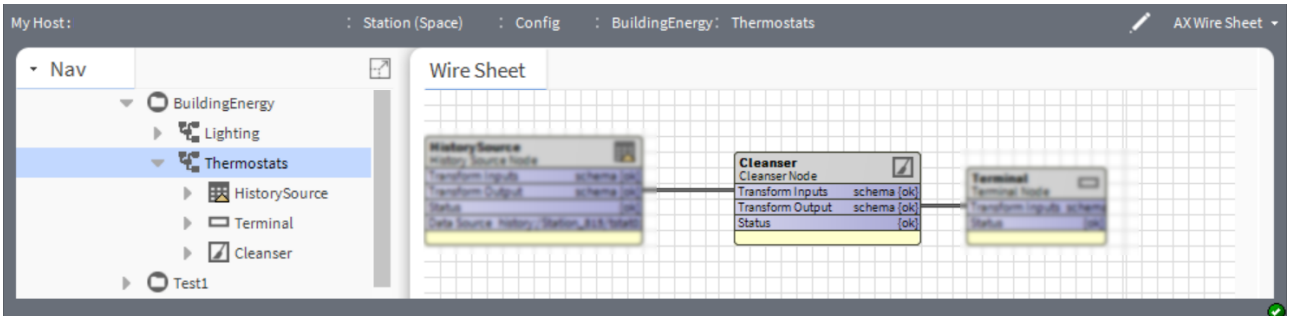
The TimeShift node works with the following data types:

- Numeric Points
- Boolean Points
- Enum Points
- String Points
  - CoV Type histories
  - Interval type histories

seriesTransform-CleanserNode

The cleanser node allows you to replace NaN values in the source data set since the NaN values cause errors while calculating the raw data from the source.

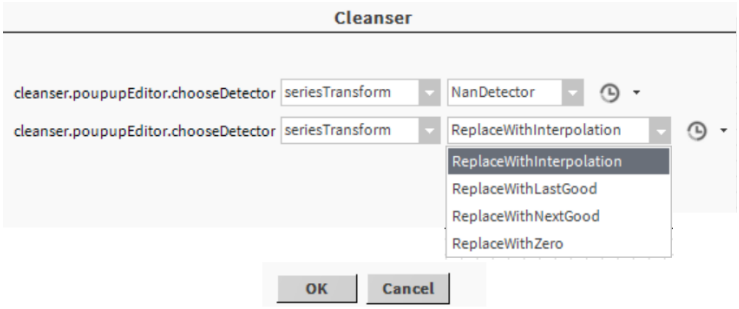
Figure 54 Cleanser Node on Wire Sheet



To use the CleanserNode component, drag it from the Series Transform palette onto the Wire Sheet view and connect a HistorySource output to the Cleanser Inputs slot. Double-click on the Cleanser component in the Wire Sheet view, and use the detectors and replacers in the **Cleanser** window to replace the NaN values in the data source.

Connect the output from the Cleanser node to another SeriesTransform node input or directly to a Terminal Node input as desired.

Figure 55 Cleanser Editor



The cleanser node have only one **NanDetector** and four replacers types.

- **ReplaceWithInterpolation**: If a NaN value is detected in the source data, replace with the average of the previous record and the next record which have valid non-NaN values
- **ReplaceWithLastGood**: If a NaN value is detected in the source data, replace with the previous record which has a valid non-NaN value.
- **ReplaceWithNextGood**: If a NaN value is detected in the source data, replace with the next record which has a valid non-NaN value.
- **ReplaceWithZero**: If a NaN value is detected in the source data, replace the NaN values with a zero value.

Resolve the graph and check that the table displays the numeric values without any NaN values.

Chapter 4 Windows

Topics covered in this chapter

- ♦ Terminal Editor
- ♦ Aggregate Editor
- ♦ Scale Editor
- ♦ Composite Editor
- ♦ Node Editor
- ♦ TimeShift window
- ♦ Filter window
- ♦ Cleanser Editor

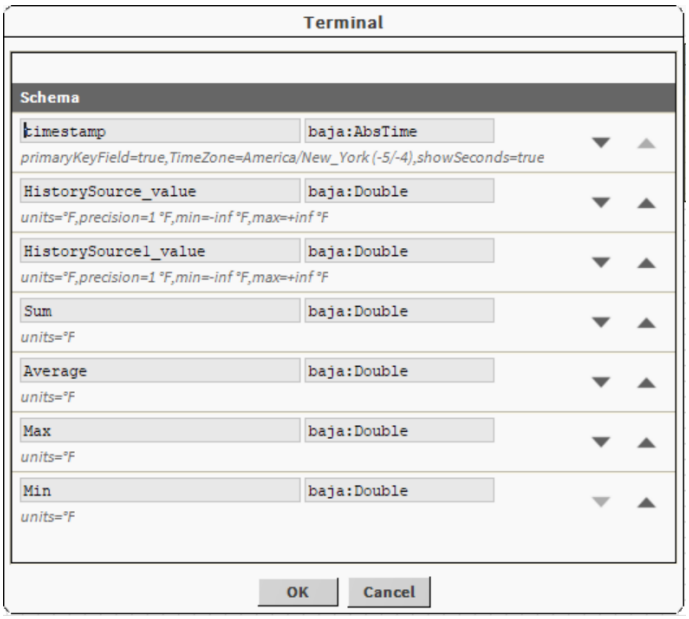
Windows create and edit database records or collect information when accessing a component. You access them by dragging a component from a palette into a station or by clicking a button.

Windows do not support **On View (F1)** and **Guide on Target** help. To learn about the information each contains, search the help system for key words.

Terminal Editor

This editor configures the final node in the series transform.

Figure 56 Terminal Editor



You open this window by ...

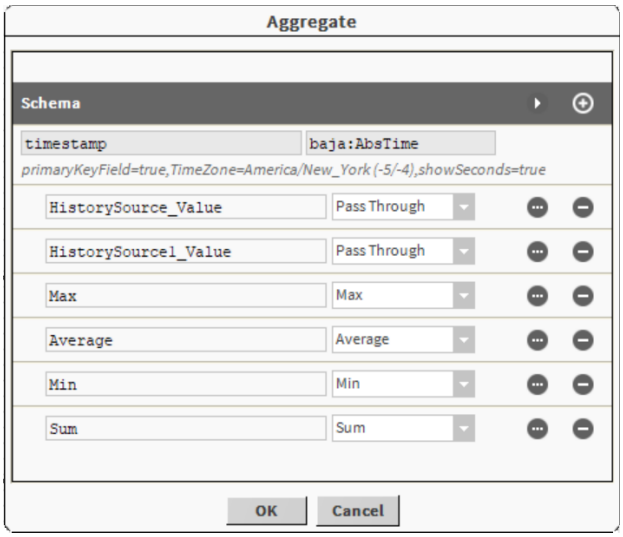
To reorder the data columns in the table view of the resolved graph data, click the up and down arrows. Columns at the top of the editor appear on the left-hand side of the table, while columns at the bottom of the editor display on the right side of the rendered table in the table view.

Aggregate Editor

The **Aggregate Editor**, like the **Composite Node** editor, creates the output schema of the aggregate node.

To create the schema, you add field names to the schema and assign functions to each field.

Figure 57 Aggregate Editor

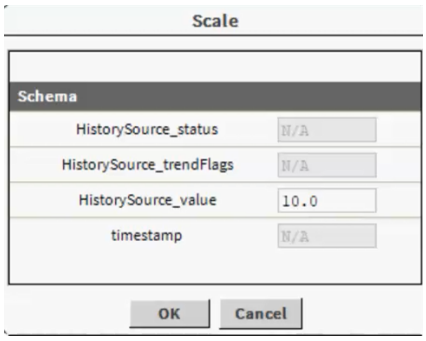


A set of source properties defines each aggregate function. You select these input properties in the **Function Editor**. There is no limit to the number of properties that you can select for a function, with the exception of the Pass Through function, which passes through a single value without altering it. This makes the value available for the next consuming node in the graph.

Scale Editor

This editor assigns a scale factor to each input.

Figure 58 Scale Editor



You open this window by ...

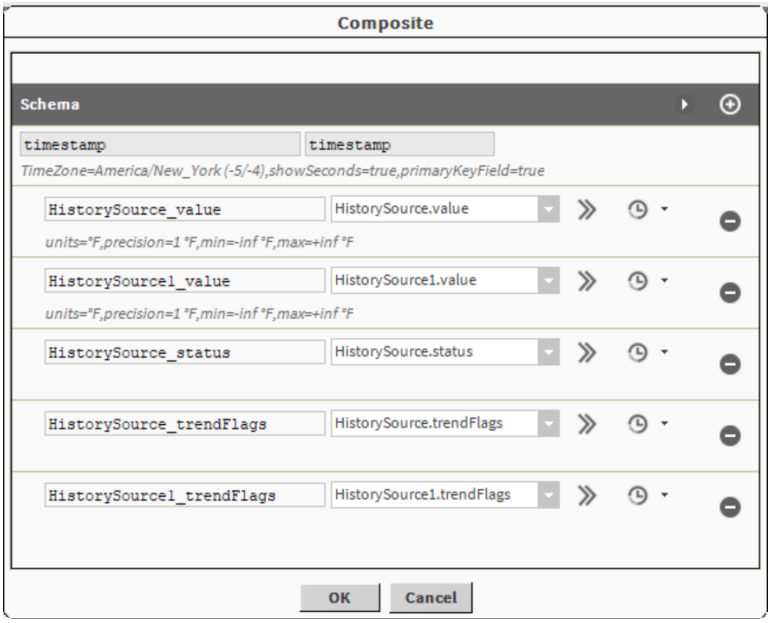
- value
- timestamp
- trendFlags
- status

Composite Editor

Composite Editor

This window configures a schema that combines multiple history sources.

Figure 59 Composite Editor



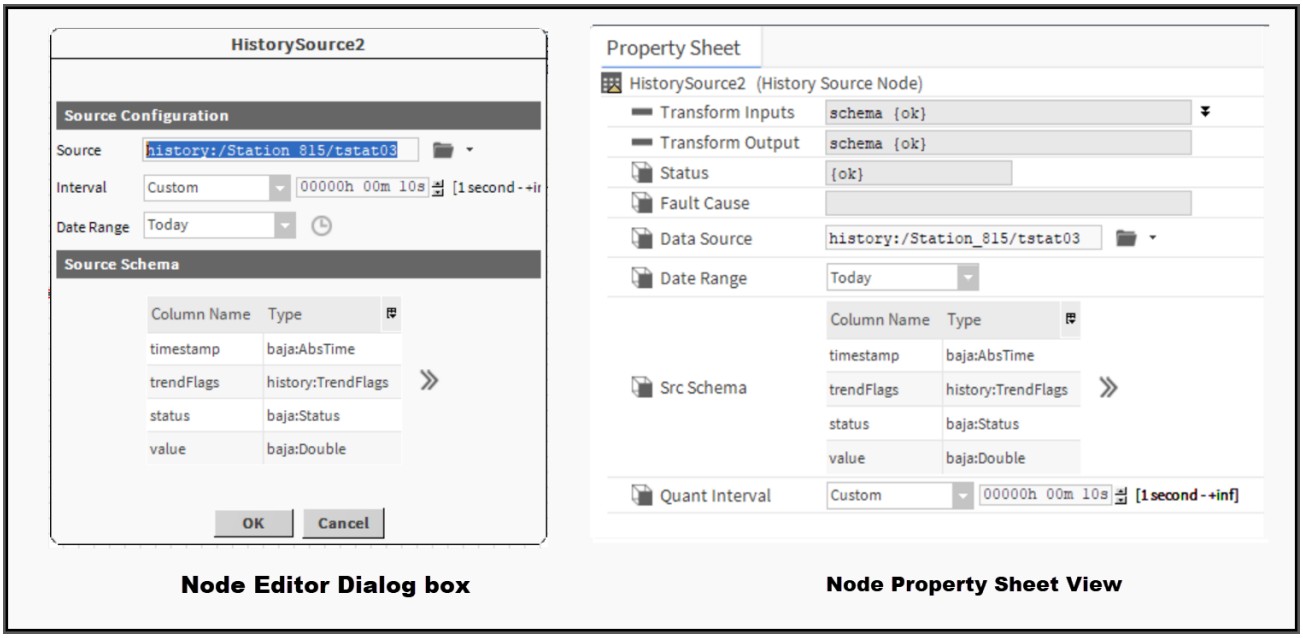
You open this window by ...

You can choose from among these values:

Node Editor

This window defines the source of the data to transform and the schema to use in the transformation.

Figure 60 Node Editor window and property sheet views



You open this window by ...

TimeShift window

This window adjusts the time values of timestamps for a generated set of data.

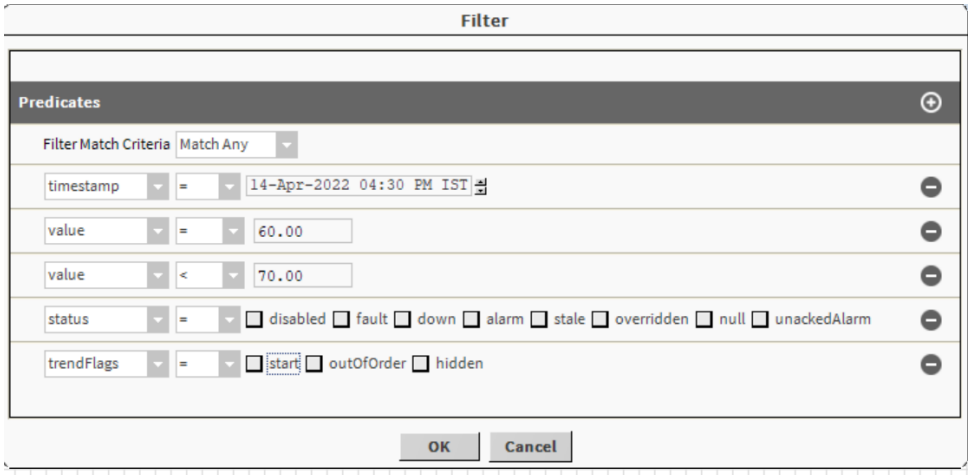
Figure 61 Set the time shift value in the TimeShift window

You open this window by ...

Filter window

This window filters a series of data using a bql predicate expression.

Figure 62 Set the Filter values in the Filters window



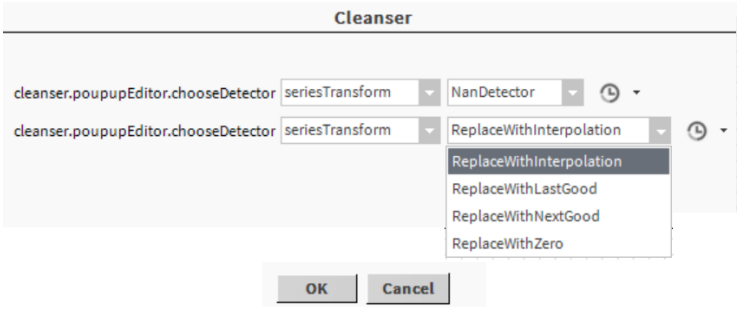
You open this window by ...

- Predicates
- Filter Match Criteria
- timestamp
- values

Cleanser Editor

The cleanser window provides a schema for replacing NaN values in the data source only for BINumeric type values.

Figure 63 Cleanser Editor



You open this window by double-click on the **Cleanser** window in the Wire Sheet view.

- NanDetector
- ReplaceWithInterpolation
- ReplaceWithLastGood
- ReplaceWithNextGood
- ReplaceWithZero

Index