

# Technical Document

## NiagaraAX Series Transform Guide

AX-3.8 and AX-3.7

November 5, 2013



# NiagaraAX-3.7/3.8 Series Transform Guide

Copyright © 2013 Tridium, Inc.

All rights reserved.

3951 Westerre Pkwy., Suite 350

Richmond

Virginia

23233

U.S.A.

## Confidentiality Notice

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 Notices

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, Niagara<sup>AX</sup> Framework, and Sedona Framework are registered trademarks, and Workbench, WorkPlace<sup>AX</sup>, and <sup>AX</sup>Supervisor, are trademarks of Tridium Inc. All other product names and services mentioned in this publication that is 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.

© Tridium, Inc. 2013. All rights reserved. The product(s) described herein may be covered by one or more U.S or foreign patents of Tridium.

# CONTENTS

<b>Preface</b> .....	<b>iii</b>
<b>Document Change Log</b> .....	<b>iii</b>
<b>Series Transforms concepts</b> .....	<b>1-1</b>
<b>Series Transform Graph and the Wire Sheet view</b> .....	<b>1-1</b>
<b>About the Series Transform Data Schema</b> .....	<b>1-1</b>
<b>Graph node configuration</b> .....	<b>1-2</b>
<b>About Graph Functions</b> .....	<b>1-3</b>
<b>Test Resolving Graph Nodes</b> .....	<b>1-4</b>
<b>About Transform Label Bindings and the cell parameter</b> .....	<b>1-4</b>
<b>Series Transform components</b> .....	<b>2-1</b>
<b>About the Transform Graph component</b> .....	<b>2-2</b>
<b>About the History Source node</b> .....	<b>2-3</b>
About Intervals and Date Range .....	2-5
<b>About the Terminal node</b> .....	<b>2-5</b>
<b>Types of Transform nodes</b> .....	<b>2-6</b>
About the Aggregate node .....	2-6
About the Composite node .....	2-7
About the Rollup node .....	2-8
About the Scale node .....	2-9
<b>Series Transform Graphs and Px views</b> .....	<b>3-1</b>
<i>Create a series transform collection table in a Px view</i> .....	3-2
<i>Create a series transform bound table in a Px view</i> .....	3-2
<i>Create a series transform chart in Px view</i> .....	3-3
<i>Create a series transform bound label in a Px view</i> .....	3-4
<b>Series Transform Component Guides</b> .....	<b>4-1</b>
<b>Components in the seriesTransform module</b> .....	<b>4-1</b>



# PREFACE

## Preface

---

[Document Change Log](#)

### Document Change Log

Updates (changes/additions) to this *Series Transform Guide* document are listed below.

- NiagaraAX-3.8 release revision, November 5, 2013.  
Changes to the document include:
  - Added sections describing the TimeShift component, see [“About the Time Shift node”](#) on page 2-10 and the FilterNode component, see [“About the Filter node”](#) on page 2-11.
  - Added sections for [TimeShift Node](#) and the [Filter Node](#).
- NiagaraAX-3.7 release revision, August 28, 2012.



# CHAPTER 1

## Series Transforms concepts

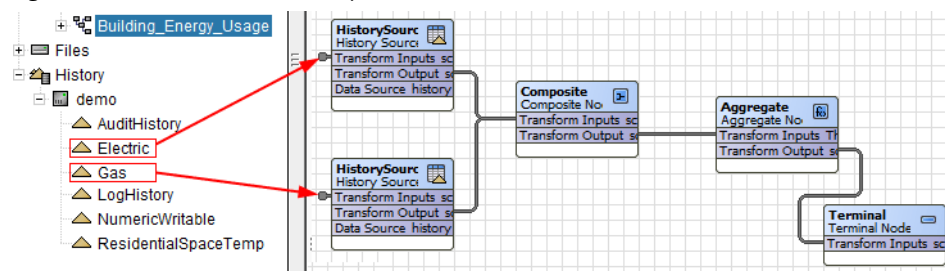
Transform Graphs provide a way to manipulate existing series data, like Niagara History records, in order to create graphs for reporting. The following sections describe some of the major concepts associated with the Series Transform Graph components.

### Series Transform Graph and the Wire Sheet view

Transform Graphs are comprised of a collection of Graph Nodes contained as properties within the parent Transform Graph component. The default view of the Transform Graph component is the wiresheet view, where you work with the various nodes to create your desired report.

Like other components, these graph nodes are interconnected (or 'wired') together using the wiresheet 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.

**Figure 1-1** Series Transform Graph Wiresheet view and source histories



Typically, you assemble Transform Graphs by interconnecting Graph Nodes in the wiresheet view of the parent Transform Graph Node, with source nodes and other nodes integrating into a final terminal node. On the wiresheet, each graph node is connected into the graph by linking the output property of the component to the input property of the target component. The source tabular data is most likely to be records that are NiagaraAX histories. Each graph produces a single table result, as represented by the single terminal node required in each graph.

### About the Series Transform Data Schema

Fundamental to the series transform graph is the concept of Data Schema. A data schema defines the structure of some piece of data. A common example of a data schema is seen in the Table view of a NiagaraAX history with several columns. For example, in the following table, the columns: **Timestamp**, **Trend Flags**, **Status**, and **Value** together are the Schema of the history.

**Figure 1-2** History table columns represent the history Schema

Timestamp	Trend Flags	Status	Value (°C)
18-May-12 1:12:16 PM EDT	{}	{ok}	59.7 °C
18-May-12 1:12:18 PM EDT	{}	{ok}	66.7 °C
18-May-12 1:12:19 PM EDT	{}	{ok}	73.6 °C
18-May-12 1:12:20 PM EDT	{}	{ok}	80.4 °C
18-May-12 1:12:21 PM EDT	{}	{ok}	87.4 °C
18-May-12 1:12:22 PM EDT	{}	{ok}	94.4 °C
18-May-12 1:12:23 PM EDT	{}	{ok}	98.7 °C
18-May-12 1:12:24 PM EDT	{}	{ok}	91.8 °C
18-May-12 1:12:25 PM EDT	{}	{ok}	84.8 °C
18-May-12 1:12:26 PM EDT	{}	{ok}	78.0 °C

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 what type of data the history represents. If the history is placed on 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 wiresheet to create an output schema for other graph nodes to use when setting up the desired 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 data type (format) of the data. Knowing the data type allows you to assign different value transformations and calculations to specific pieces of the data.

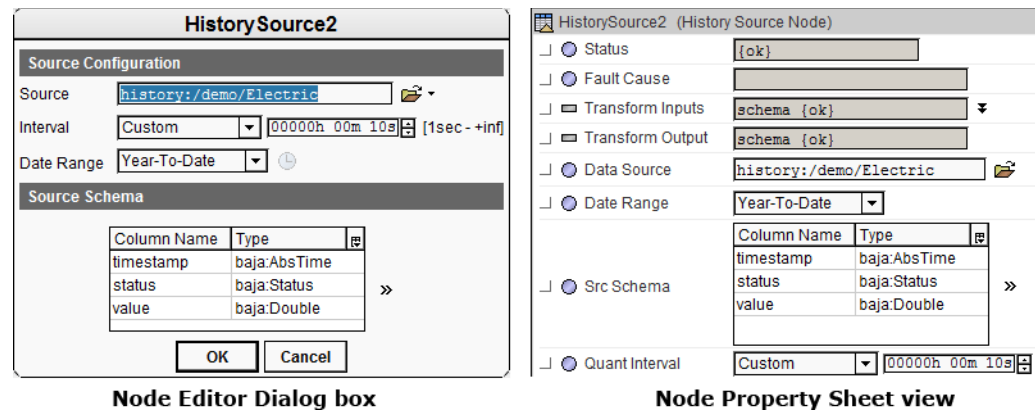
As an example, imagine that you want to scale the value of a history record by a factor of 3.9. How would you know that the **Value** column *can* be scaled? If the **Value** column contains Boolean data, you cannot scale the data by number (what is 3.9 times False?). By knowing both the name and the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

## Graph node configuration

You can configure a graph node from its property sheet view, however, it is much easier to use the associated dialog box editor of the graph node. To open the appropriate dialog box, double click the component in the wiresheet view. These editors are the best way to configure the graph node, however, the Property Sheet view is still available for each graph.

**Figure 1-3** Node Editor dialog box and property sheet views

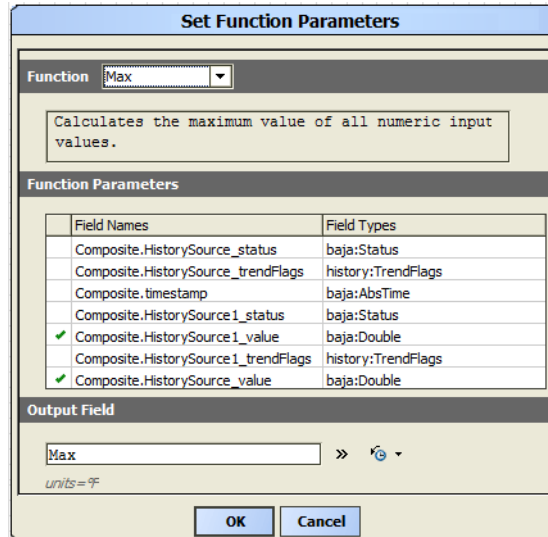




## About Graph Functions

Aggregate nodes and Rollup nodes can apply functions, such as Sum, Average, Max, and others, to the output. The Function Editor provides a way to configure functions assigned to the output schema field of either type of node.

**Figure 1-4** Function Editor



The function editor consists of three main sections:

- **Function selector**  
The function selector allows you to select which function to assign to an output field. All available functions are contained in the function option list. Each function includes a function description.
- **Parameter selector**  
The function parameter selector is a table of fields available as argument data sources. Each row of the table represents a schema field from an incoming schema 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 1-5** Function Parameters

Field Names	Field Types
Composite.HistorySource_status	baja:Status
Composite.HistorySource_trendFlags	history:TrendFlags
Composite.timestamp	baja:AbsTime
Composite.HistorySource1_status	baja:Status
✓ Composite.HistorySource1_value	baja:Double
Composite.HistorySource1_trendFlags	history:TrendFlags
✓ Composite.HistorySource_value	baja:Double

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

- **Field Assignment editor**  
When a function executes, it assigns the results to the associated node schema output field. You can type the field name into the output field text box editor. The function selection determines the field data type (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 that you wish to use as data arguments for a Max function. The first field values are defined in units of Celsius, while the second field units are defined in units of Fahrenheit. In order 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 1-6** Output field

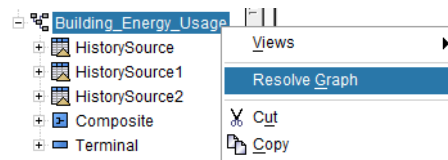


By assigning the unit type to the output field, you provide the common unit type used for comparison by the function at time of cursor execution. You also determine the unit type used for the comparison result, which determines the actual value of the result data.

## Test Resolving Graph Nodes

Each Transform Graph Node includes a **Resolve Graph** menu option in the popup menu of the transform graph component. For example, select this menu item to test your graph from the workbench so you know how it's working 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 1-7** Resolve Graph menu item



The resolved data is displayed in the workbench table view and can be used to see if you get the results expected from your configuration. If the results are not as expected, you can make changes and resolve the graph again.

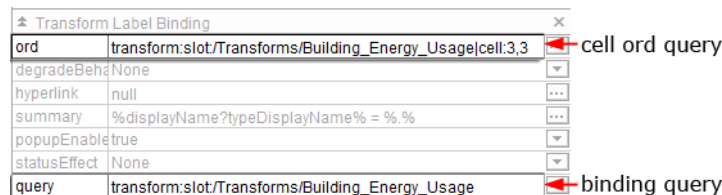
**Figure 1-8** Transform Graph resolved in collection table

timestamp	HistorySource_value (°F)	HistorySource1_value (°C)	Sum (°F)	Average (°F)	Max (°F)	Min (°F)
21-May-12 3:38:00 PM EDT	350.6 °F	58.1 °C	11693.08	243.61	350.56	136.65
21-May-12 3:39:00 PM EDT	350.4 °F	57.8 °C	28217.54	243.25	350.40	136.11
21-May-12 3:40:00 PM EDT	350.7 °F	55.7 °C	28006.92	241.44	350.68	132.20
21-May-12 3:41:00 PM EDT	350.7 °F	61.1 °C	28080.15	246.32	350.67	141.97
21-May-12 3:42:00 PM EDT	350.8 °F	60.1 °C	28475.06	245.47	350.83	140.12
21-May-12 3:43:00 PM EDT	350.6 °F	58.7 °C	28320.12	244.14	350.57	137.71
21-May-12 3:44:00 PM EDT	350.4 °F	57.7 °C	28201.82	243.12	350.38	135.86
21-May-12 3:45:00 PM EDT	350.8 °F	55.9 °C	28043.38	241.75	350.80	132.70
21-May-12 3:46:00 PM EDT	350.3 °F	61.7 °C	28119.13	246.66	350.29	143.02
21-May-12 3:47:00 PM EDT	350.3 °F	28.3 °C	6065.47	216.62	350.34	82.90

## About Transform Label Bindings and the cell parameter

The Transform Label Binding is similar to other bound label bindings but it has an additional query property and an ord query which allows you to designate a specific table cell.

**Figure 1-9** Transform Label Binding



- Binding query**  
 The query property of the transform binding is used to override select runtime properties of the graph nodes which 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.

- **Cell ord query**

The cell ord query allows you to specify column and row in the table that the selected Transform-Graph 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 indices, for example 0,0 specifies the cell defined by the first column and first row in a table.

**Figure 1-10** Bound label displays text from specified table cell

**Bound Label Example Using Cell Designation**

Time	BldgAKW	BldgBkW	BldgCkW
29-Feb-12 8:28 AM EST	1767.36	1845.18	1507.08

**Bound Label Properties**

ord: station:|slot:/SeriesTransGraphs/BuildingEnergy/transform|cell:3,0

**Resolved Graph Collection Table**

Timestamp	BldgAValue (kW-hr)	BldgBValue (kW-hr)	BldgCValue (kW-hr)
29-Feb-12 8:28:20 AM EST	1767.4 kW-hr	1845.2 kW-hr	1507.1 kW-hr
29-Feb-12 8:28:25 AM EST	1917.3 kW-hr	1801.9 kW-hr	1425.1 kW-hr
29-Feb-12 8:28:30 AM EST	2074.6 kW-hr	1718.4 kW-hr	1402.0 kW-hr



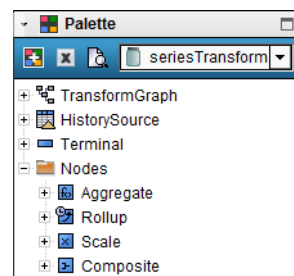
# CHAPTER 2

## Series Transform components

Additional nodes may be created by third-party developers using the series transform API. However, with NiagaraAX-3.7, there are seven individual objects provided on the seriesTransform palette. The following components can be thought of as comprising four general categories of Series Transform objects:

- **Container (Transform Graph node)**  
The TransformGraph object 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 as desired.
- **Source (HistorySource node)**  
The HistorySource node is the single object available for identifying the source data for your transformation.
- **Terminal (Terminal node)**  
There is one terminal node that is the final node of any transform graph. Each Transform Graph must have one and only one terminal node.
- **Transforms (Transform nodes)**  
Transform nodes provided on the palette include the Aggregate, Composite, Rollup, and Scale nodes.

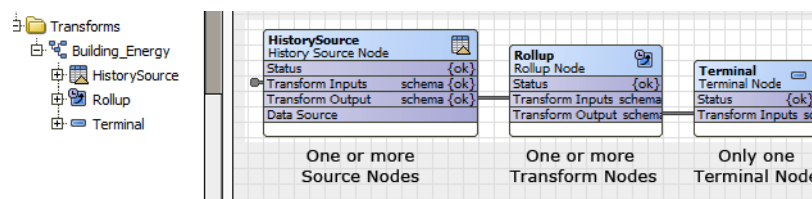
**Figure 2-1** Series Transform Graph palette



By adding components from the seriesTransform palette onto a Transform Graph wiresheet view, you can create a static definition of how run-time data is transformed into a new single-output set of information. The transform graph is created 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 flows from one transformation node to the next.

**Note:** All graph nodes must be contained by a Transform Graph container. All nodes within the Transform Graph container must be uniquely named.

**Figure 2-2** Minimum set of nodes on the seriesTransformGraph wiresheet view



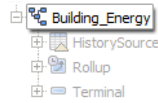
Each of the graph nodes described in the following sections includes a special popup editor that displays when clicking the graph node component in the wiresheet view. The Property Sheet view is still accessible, but must be selected from the workbench View Selector.

## About the Transform Graph component

A Series Transform Graph is a node container that can hold all other graph nodes. The transform graph node itself is also a graph node, making it possible for transform graphs to contain other transform graphs. This allows transform graphs to be created to fulfill very specific data functionality and used as building blocks to create larger graphs.

After you add a Transform Graph node to your station, you can double-click on it to display the node's default wiresheet view for configuration of the graph. When properly configured, the Series Transform Graph performs a transformation or series of transformations against a set of data to produce a single result.

**Figure 2-3** Transform Graph component

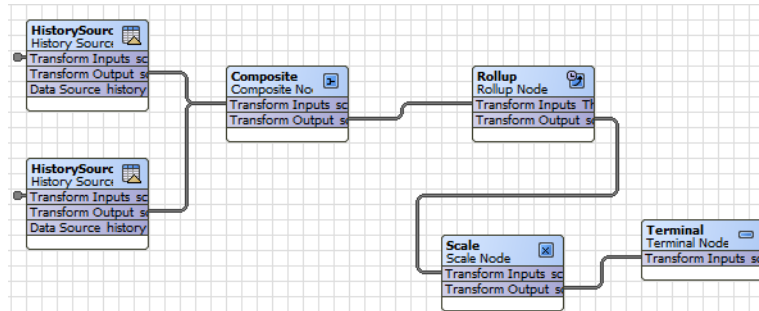


Note the following summary information about Transform Graph Nodes:

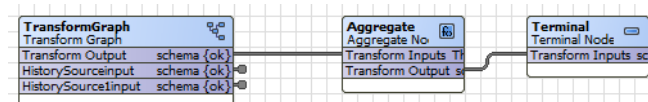
- Transform Graph nodes must contain a single Terminal node plus one or more source nodes to define the schema of the data. Source nodes may be HistorySource nodes or other Transform Graph nodes.
- Each Transform Graph node returns a single result and thus contains a single Terminal node.
- Transform Graph nodes may contain one or more Transformation-type nodes, 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 (refer to “Series Transform components” on page 2-1).

The following illustrations show some example Transform Graph implementations.

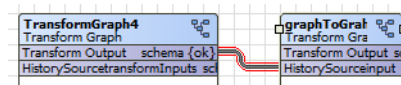
**Figure 2-4** A graph containing graph nodes



**Figure 2-5** A graph containing another graph as a source



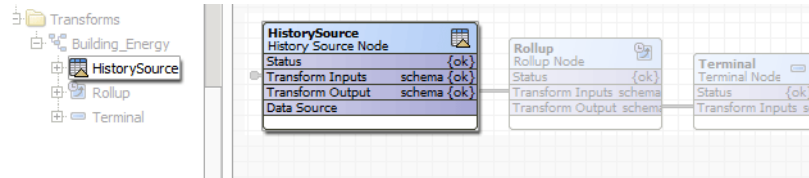
**Figure 2-6** Two interconnected graphs. The first graph acts as data source for the second.



## About the History Source node

The HistorySource node represents the source data (most often a NiagaraAX history) and defines the data schema for data that is processed by the Transform Graph. Unlike other graph nodes, this graph node does not have an input property displayed in the wiresheet. This is because the node itself is a data source and will use the node configuration to obtain the source data. The HistorySource node should be placed on a Transform Graph node.

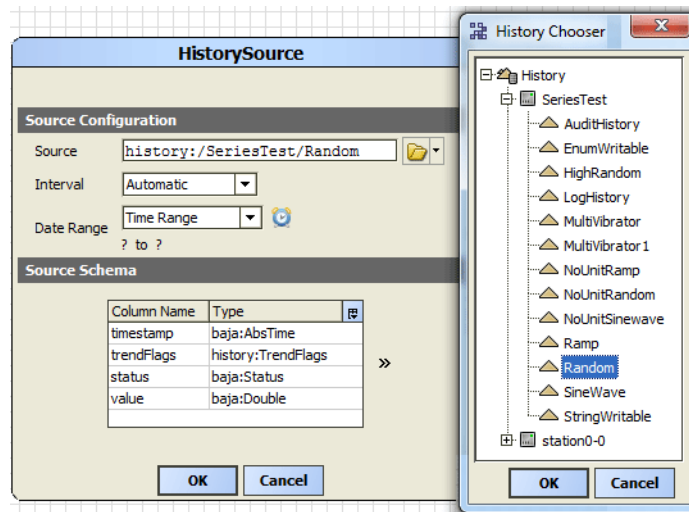
**Figure 2-7** HistorySource node



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

To use the HistorySource node, select the history source ORD from the HistorySource Source property. Select the Source ORD by clicking the folder icon next to the text field. The source ord must point to a data series. Usually this is limited to a history ord. You can also type the ORD path directly in the text field.

**Figure 2-8** History Source ORD

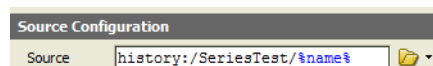



When the Source value is set to a valid history ORD, the Source Schema table is automatically populated with the schema of the ORD's history schema. In Figure 2-8 the schema is the same as the column names in the corresponding source History Table view. Beside each column name is the data type that the column represents.

When you set the HistorySource ORD you can also take advantage of the following features:

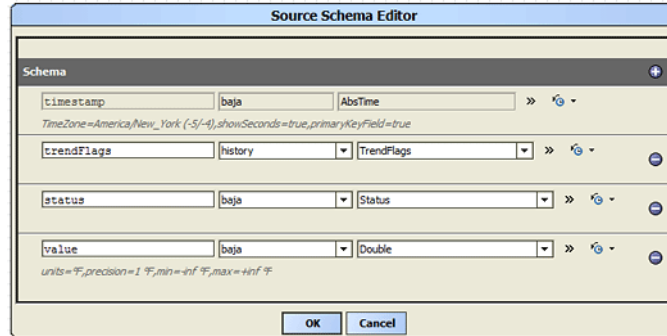
- Parameterized History ORD**  
 The source ORD uses the BFormat syntax at time of graph resolution. This allows the source ORD to be parameterized. This is especially useful when creating a transform graph that will be displayed in a Px file. The history ORD can be set using BFormat syntax to create a transform graph that can be used in many Px files, or the same Px file set as a view on different components. In the latter case, the BFormat syntax resolves against the component that contains that Px view.

**Figure 2-9** Source Configuration field



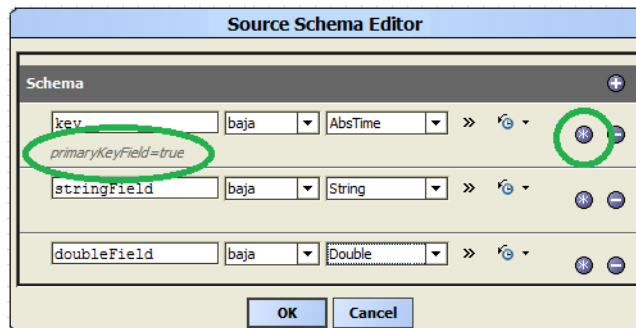
- Just In Time Source ORD**  
 The transform graph can assign values to attributes of a graph node at the time the graph is resolved. In the case of the HistorySource node, the history ORD for the data source can be given at resolve time.  
 If a Source ORD is not set at design time, a Source Schema must be defined for the node. The Source Schema is the schema of the data that history source represents. This schema data is used by other graph nodes to create instructions on how to manipulate the data. The schema of a History Source node can be manually configured using the Schema Source Editor. Click the chevron button  next to the Source Schema table to display the editor.

**Figure 2-10** Source Schema Editor



Configure the schema by setting a string field name value and then define the data type for that field value. You can think of a field value as a column of data. The data type is a baja “Type” that you can define by selecting (from the first option list) the module that the data type belongs to (usually baja) and then (from the second option list) the specific type in the module that the column represents. Along with the name and data type information, each schema field can be configured with facet information. This information will tag the data column with important information such as the unit type associated with the value of the column (for example, Fahrenheit). If you design your own schema, you must select the key field of the schema. Each schema includes a key field. The key field is the data field that will include a unique value for each data row at the time that the graph node cursor is resolved. The key field is similar to the primary key field used in relational databases.

**Figure 2-11** Source Schema Editor



In the case of histories, the key field is the Timestamp field. Each record returned by a history has a unique Timestamp value. By using the Timestamp field as the key field, the transform framework can perform tasks such as grouping a collection of records together in five minute intervals. This is possible because key field allows the framework to uniquely identify each cursor record, or “table row”, as a unique record.

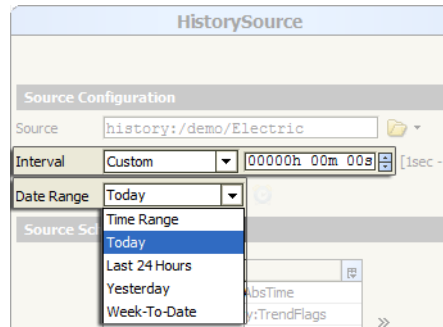


### About Intervals and Date Range

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 coalesce 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** dialog box provides fields that allow you to manually set intervals to predefined increments or choose predefined increments using the **Interval** option list.

Figure 2-12 Interval and Date Range fields



The interval can also be set to **Automatic** so that the current interval of the history collection is used. For example, if the history is already set to collect at 15 minute intervals, the automatic quantization interval of the history source node is also set to 15 minutes.

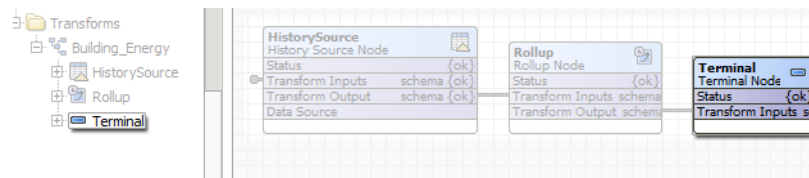
**Note:** It is important to note that all values are quantized based on a starting time of 0:0:0.000. This means that an interval of 15 minutes will quantize a record whose timestamp reflects the value of 11:13:52.093 to 11:00:00.000.

Often it is desirable to use only a segment of a History rather than the entire history for a point value. Use the Date Range property to select the starting and ending date and times for the desired history data. All history records between these date values, including those histories which fall exactly on those date values, will be included in the data set at the time the graph is resolved.

### About the Terminal node

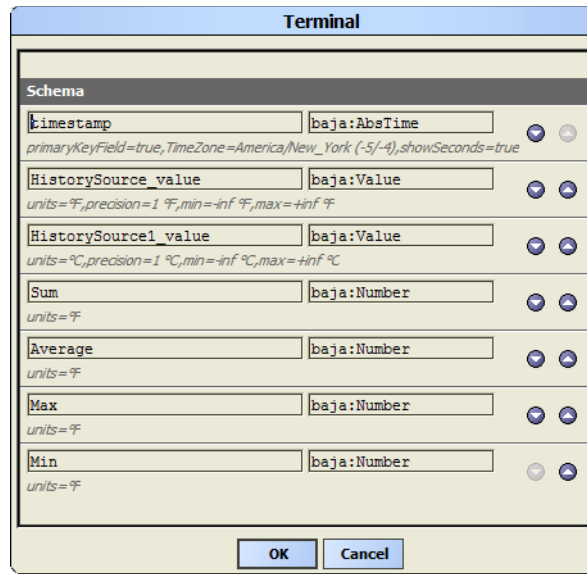
The Terminal node is used to designate the result schema of a Transform Graph. Each graph must include one and only one terminal node. The Terminal editor allows the schema fields (data columns) of the graph result to be reordered. This is especially useful when using the Resolve Graph nav menu option of a transform graph to test the graph result.

Figure 2-13 Terminal node



The **Terminal Editor** allows the schema fields (data columns) of the graph result to be reordered. This is especially useful when using the **Resolve Graph** nav menu option of a transform graph to test the graph result.

**Figure 2-14** Terminal Editor

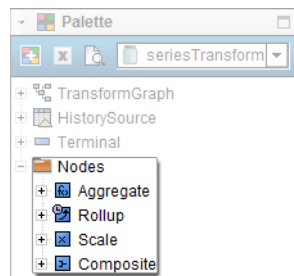


To reorder the data columns in the table view of the resolved graph data, click the up and down arrows. Columns at the top of the editor will appear on the left hand side of the table, while columns at the bottom of the editor will display on the right side of the rendered table in the table view.

## Types of Transform nodes

Transform nodes are located in the Nodes folder of the seriesTransform palette and include the following:

**Figure 2-15** Transform nodes located in the seriesTransform palette

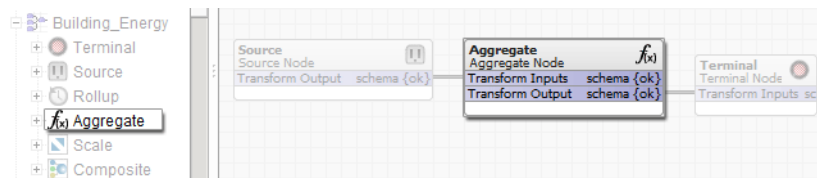


- Aggregate node, see “About the Aggregate node” on page 2-6 for details.
- Composite node, see “About the Composite node” on page 2-7 for details.
- Rollup node, see “About the Rollup node” on page 2-8 for details.
- Scale node, see “About the Scale node” on page 2-9 for details.
- TimeShift node, see “About the Time Shift node” on page 2-10 for details.
- Filter node, see “About the Filter node” on page 2-11 for details.

### About the Aggregate node

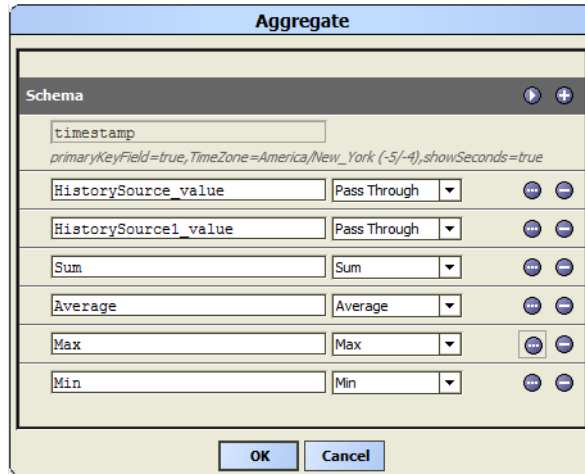
The Aggregate node is used to perform functional operations against each row of data produced by the node data source. These functional operations do not take into account the order of the parameters, and so are limited to functions dealing with input values as an aggregate.

**Figure 2-16** Aggregate node



The **Aggregate Editor**, like the **Composite Node** editor, is used to create the output schema of the aggregate node. The schema is created by adding field names to the schema and assigning functions to each field.

**Figure 2-17** Aggregate Editor

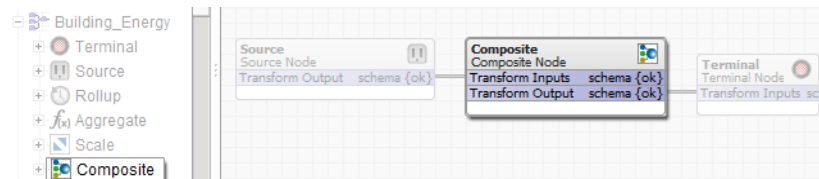


Each aggregate function takes a set of input source fields as parameter values. You select these input parameters in the **Function Editor** (see “About Graph Functions” on page 1-3). There is no limit to the number of parameters that you can select for a function, with the exception of the **Pass Through** function, which passes through a single value without altering the value, making the value available for the next consuming node in the graph.

### About the Composite node

The Composite node allows you to combine more than one input source into a single data structure. Fields of input sources can be included multiple times into the composite schema so long as they are uniquely named each time. In addition to using multiple **HistorySource** nodes you can include a single incoming schema element multiple times in the Composite node.

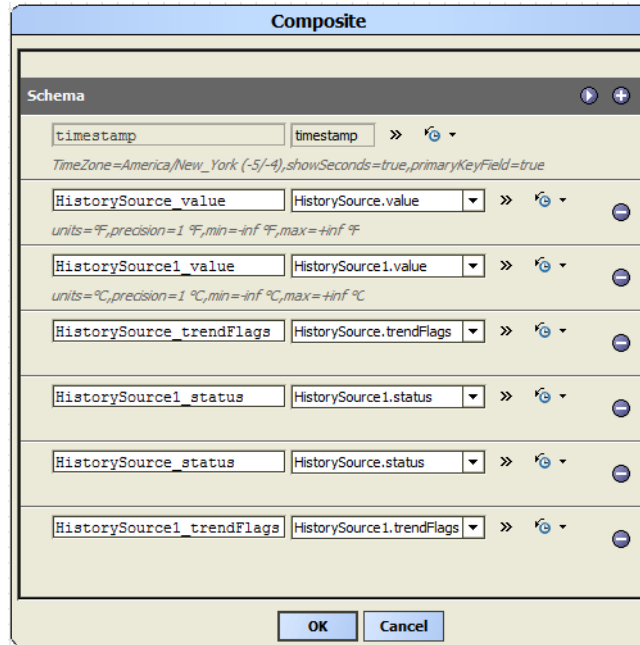
**Figure 2-18** Composite node





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 schemas.

**Figure 2-19** Composite Editor



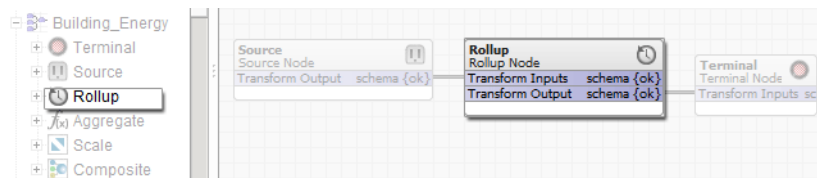
When you click the **Add** button , an empty text field and option list display in the editor. To complete the addition of a field, first type a unique name in the text field for the schema field and then select an input field from the drop down list. Each field in the option list is a field from one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

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

### About the Rollup node

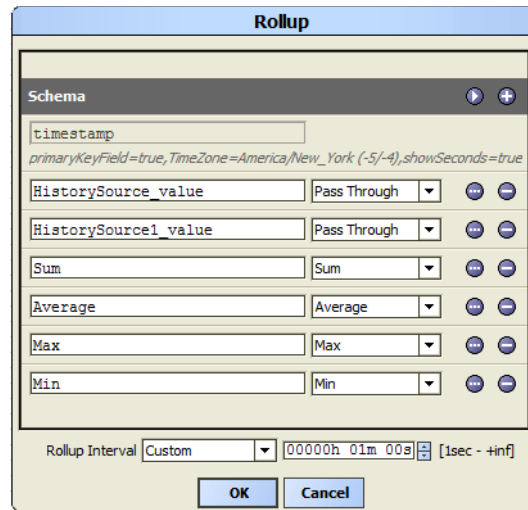
Like the **Aggregate** node, the **Rollup** node can perform functional operations against each row of data produced by linked data sources. Unlike the **Aggregate** node, the **Rollup** node performs operations against several rows of data. The number of rows used to collect function argument data is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


**Figure 2-20** 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 2-21** Rollup Editor



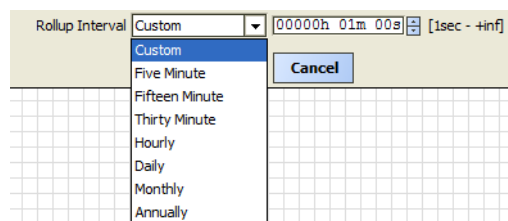
When you click the **Add** button , an empty text field and drop down list display in the editor. To complete the addition of a field, first type a unique name in the text field for the schema field and then select an input field from the drop down list. Each field in the drop down list is a field from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

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

Each function takes a set of input source schema fields as parameter values. You can use the Function Editor to select parameters to apply. There is no limit to the number of parameters that may be selected for a function, with the exception of the Pass Through function, which passes through a single value without altering the value, making 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.

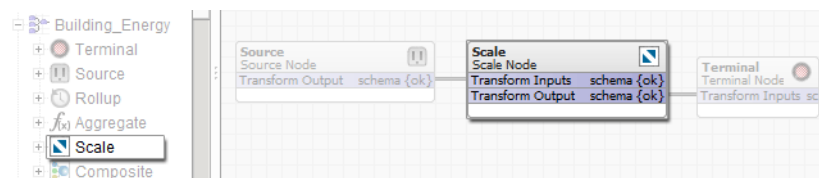
**Figure 2-22** Rollup Interval selector



### About the Scale node

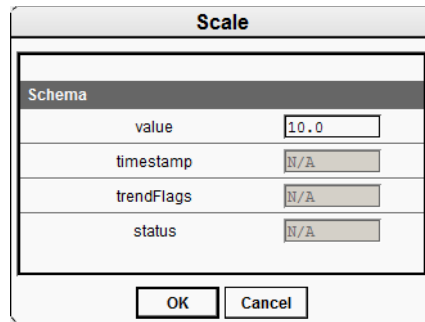
The Scale node lets you multiply selected numeric input fields from a HistorySource node to create an Output that is the product of the Input and the Scale factor. You cannot modify non-numeric data (such as Status). The “scaled” output can then be linked to the Input of a terminal.

**Figure 2-23** Scale node



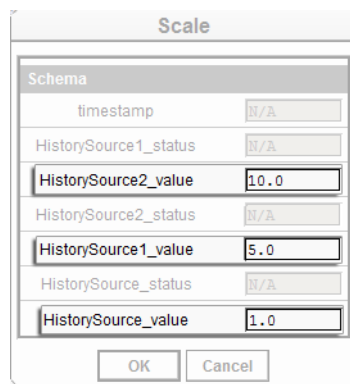
The Scale Editor lets you add a scale value in the value field for any numeric data.

**Figure 2-24** Scale Editor



You can use the **Scale Editor** to assign scale factors for each input. For example, in the following illustration, scale factors are added 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 2-25** Scale Editor with three inputs

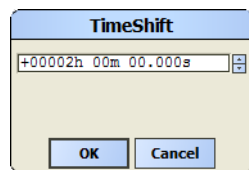


### About the Time Shift node

The TimeShift node lets you adjust the time values of timestamps on a set of data that is generated to a SeriesTransformTable (using the TransformGraph component).

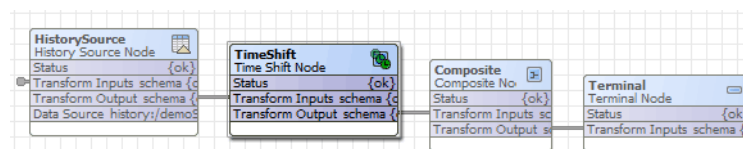
To use the TimeShift node, drag it from the Series Transform palette onto the wiresheet view and connect a HistorySource output to the TimeShift Inputs slot. Double-click on the TimeShift component in the wiresheet view and set the desired time shift value in the **TimeShift** dialog box.

**Figure 2-26** Set the time shift value in the TimeShift dialog box



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

**Figure 2-27** TimeSheet node on wiresheet



Resolve the graph and check that the table displays time values that are offset by the value that you set in your **TimeShift** dialog box. So, for example, if you shifted your time by setting a 30 minute value in the TimeShift dialog box, then 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:

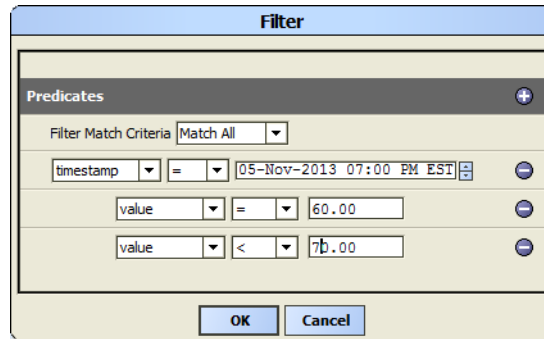
- Numeric Points
- Boolean Points
- Enum Points
- String Points
  - COV Type histories
  - Interval type histories

### About the Filter node

The Filter node (BqlFilter Node) allows you to filter 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.

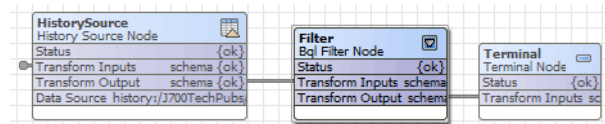
To use the FilterNode component, drag it from the Series Transform palette onto the wiresheet view and connect a HistorySource output to the Filter Inputs slot. Double-click on the Filter component in the wiresheet view and use the **Filter** dialog box to create a custom bql query.

**Figure 2-28** Set the Filter values in the Filters dialog box



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

**Figure 2-29** Filter node on the wiresheet view



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



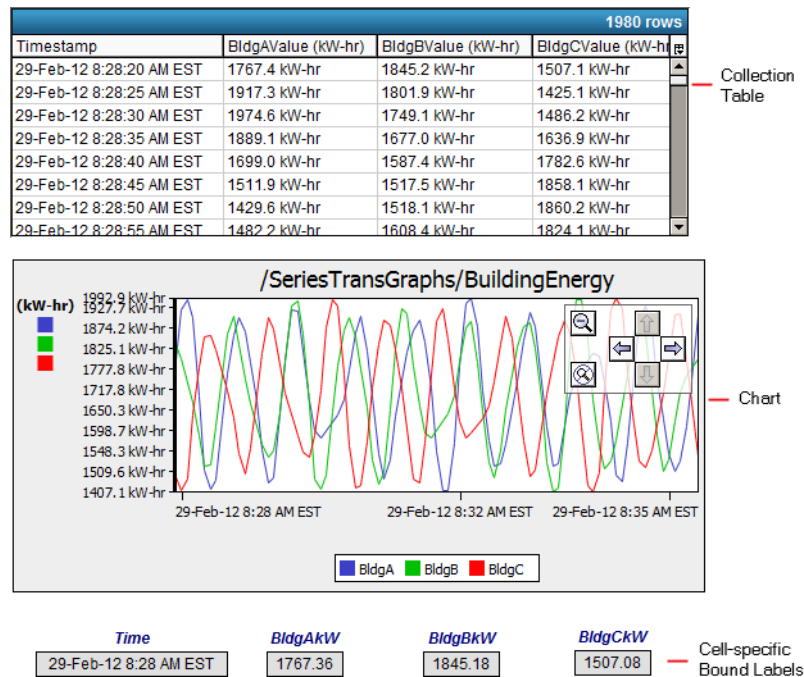


# CHAPTER 3

## Series Transform Graphs and Px views

You can create Px page views with data from the seriesTransform component. Figure 3-1 shows a single Px page that uses three different widgets to display the same resolved Series Transform Graph.

**Figure 3-1** Series transform graph data in Px view (Collection Table, Chart, Bound Labels)



The following sections describe how to use a Series Transform Graph in a Px view.

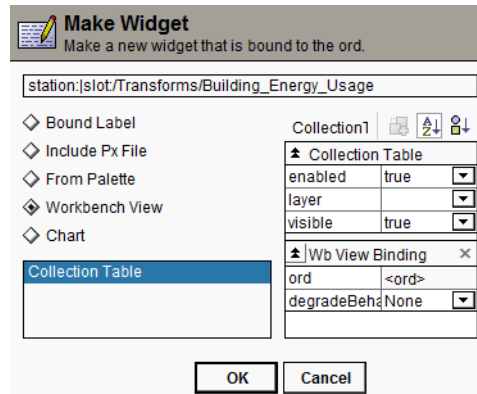
- Create a series transform collection table in a Px view
- Create a series transform bound table in a Px view
- Create a series transform chart in Px view
- Create a series transform bound label in a Px view

### Create 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 3-2** 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.

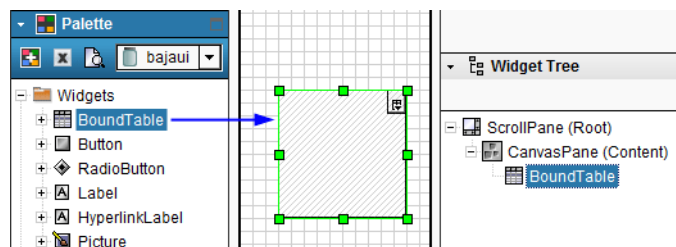
The Collection Table view displays in the Px view.

### Create 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.

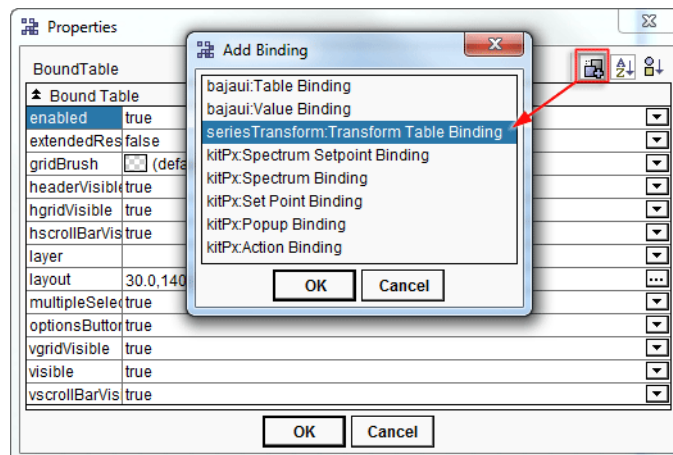
- Step 1 Drag a BoundTable widget from the bajui palette onto your Px page in the Px Editor view.


**Figure 3-3** Add BoundTable from bajui palette



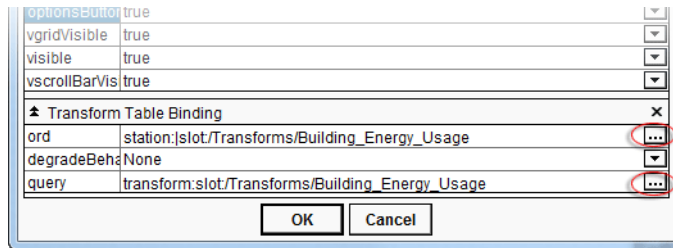
- Step 2 Double-click on the BoundTable component in the WidgetTree palette to open the Properties dialog box.


**Figure 3-4** Add Transform Table Binding



- Step 3 Click the **Add Binding** button on the **Properties** dialog box, choose `seriesTransform:Transform Table Binding` from the **Add Binding** dialog box and click **OK** in the **Add Binding** dialog box only.  
This adds the Transform Table Binding to the bottom of the **Properties** dialog box.
- Step 4 In the Transform Table Binding rows of the **Properties** dialog box, click the edit button  on the `ord` row.  
The **ord** dialog box opens.

**Figure 3-5** Add Transform Table Binding



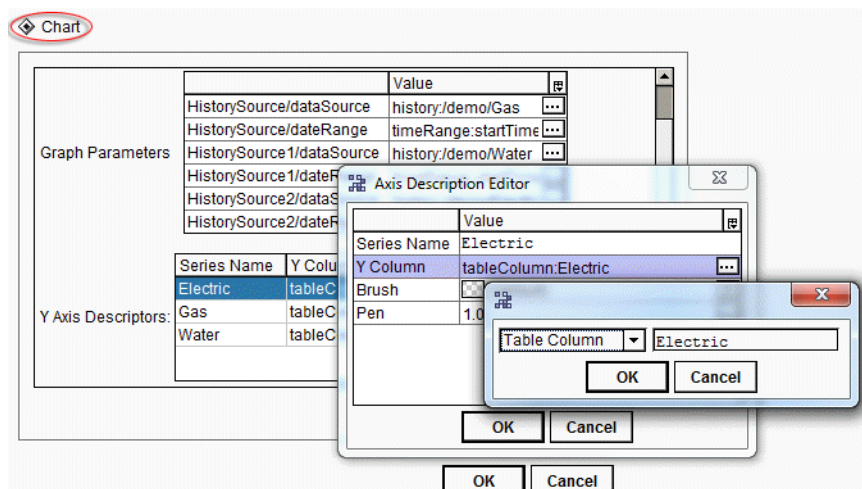
- Step 5 In the **ord** dialog box, use the Component Chooser option to browse to the desired Transform Graph component, select the component and click the **OK** button.  
The **ord** binding is set.
- Step 6 In the Transform Table Binding rows of the **Properties** dialog box, click the **Edit** button  on the `query` row.  
The **Select Ord** dialog box opens.
- Step 7 In the **Select Ord** dialog box, 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 dialog boxes to complete the setup and display the Bound Table on Px page.

### Create 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 3-6** 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.*

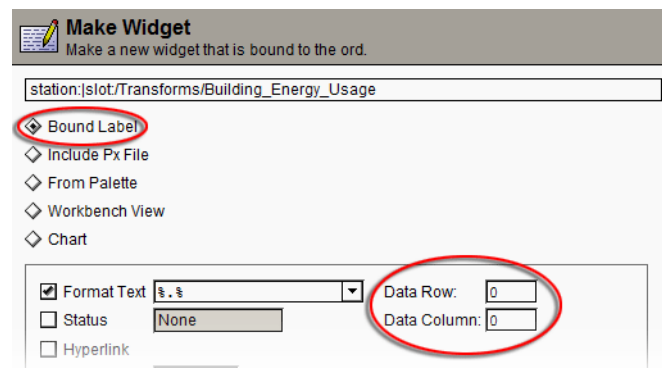
Step 4 After adding all the Y columns, click the **OK** buttons to close all the dialog boxes and complete the chart.

### Create 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 3-7** Adding the row and column to identify the table cell



The bound label displays on the Px page.

# CHAPTER 4

## Series Transform Component Guides


---

These component guides provide summary help on NiagaraAX Series Transform components.


### Components in the seriesTransform module

- [Series Transform Graph](#)
- [History Source Node](#)
- [Series Schema](#)
- [Series Interval](#)
- [seriesTransform-TerminalNode](#)
- [Terminal Map](#)
- [Aggregate Node](#)
- [RollupNode](#)
- [Function Map](#)
- [Rollup Interval](#)
- [Rollup Interval](#)
- [Scale Node](#)
- [Scale Factors](#)
- [Composite Node](#)
- [Composite Map](#)
- [TimeShift Node](#)
- [Filter Node](#)


#### **seriesTransform-TransformGraph**

 The TransformGraph object 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 as desired. See [“Series Transform Graph and the Wire Sheet view”](#) on page 1-1 for more details.


#### **seriesTransform-HistorySourceNode**

 The HistorySource node is the single object available for identifying the source data for your transformation. See [“About the History Source node”](#) on page 2-3 for more details.


#### **seriesTransform-SeriesSchema**

 The Data Schema concept is fundamental to the series transform graph. A data schema defines the structure of some piece of data. See [“About the Series Transform Data Schema”](#) on page 1-1 for more details.

#### **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. See [“About Intervals and Date Range”](#) on page 2-5 for more details.

#### **seriesTransform-TerminalNode**

 There is one terminal node that is the final node of any transform graph. Each Transform Graph must have one and only one terminal node. See [“About the Terminal node”](#) on page 2-5 for more details.

**seriesTransform-TerminalMap**

■ The TerminalMap component represents the mapping of source node values to the terminal. You can use the Terminal Editor to arrange the order these properties, if desired. See [“About the Terminal node”](#) on page 2-5 for more details.

**seriesTransform-AggregateNode**

■ The Aggregate node performs functional operations against each row of data produced by the node data source. See [“About the Aggregate node”](#) on page 2-6 for more details.

**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. See [“About the Rollup node”](#) on page 2-8 for more details.

**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. See [“About the Aggregate node”](#) on page 2-6 and [“About the Rollup node”](#) on page 2-8 for more details.

**seriesTransform-RollupInterval**

■ The Rollup Interval component holds the exact (custom or default option) time intervals that you select for a particular Rollup node. See [“About the Rollup node”](#) on page 2-8 for more details.

**seriesTransform-ScaleNode**

■ The Scale node lets you multiply selected numeric input fields from a HistorySource node to create an Output that is the product of the Input and the Scale factors. See [“About the Scale node”](#) on page 2-9 for more details.

**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. See [“About the Scale node”](#) on page 2-9 for more details.

**seriesTransform-CompositeNode**

■ The Composite node allows you to combine more than one input source into a single data structure. Fields of input sources can be included multiple times into the composite schema so long as they are uniquely named each time. See [“About the Composite node”](#) on page 2-7 for more details.

**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. See [“About the Composite node”](#) on page 2-7 for more details.

**seriesTransform-TimeShift Node**

■ The Transform node provides a way for you to adjust the timestamp values that are resolved into a series transform table. The Time Shift node component is available on the Series Transform palette. See [“About the Time Shift node”](#) on page 2-10 for more details.

**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. See [“About the Filter node”](#) on page 2-11 for more details.