

Dataflow Editor User Guide

Kinetic - Edge & Fog Processing Module (EFM) 1.7.0

Revised: July 18, 2019

Table of Contents

Conventions	7
User Interface Icons	7
Dataflow Editor overview	8
Structure of the Dataflow Editor	8
Opening a dataflow model	9
Dataflow blocks.....	10
<i>Adding a block</i>	11
<i>Selecting a block</i>	12
<i>Renaming a block</i>	13
<i>Moving a block</i>	14
<i>Deleting blocks</i>	14
<i>Grouping blocks</i>	14
<i>Ungrouping blocks</i>	14
Block properties.....	15
<i>Adding and removing Input/Output properties</i>	15
<i>Pinning and unpinning properties</i>	15
<i>Manually setting property values</i>	16
<i>Viewing Help for a property</i>	17
Bindings.....	17
<i>Using arrowheads to create bindings to block properties</i>	17
<i>Creating bindings to and from the Block Properties pane</i>	18
<i>Event and trigger properties</i>	19
<i>Finding the valid string to bind to an enum property</i>	20
<i>Using Dataflow Editor syntax to review or edit a binding</i>	21
<i>Using the arrowhead to delete a binding</i>	21
<i>Using the Block Properties pane to delete a binding</i>	21
<i>Deleting and resetting a binding</i>	22
<i>Copying and pasting a path or property value</i>	22
<i>Using the small Binding popup window</i>	22
Tables	24
Loading a table.....	24

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Conventions

<i>Loading a table from a metric history</i>	24
<i>Loading a table from multiple metric histories</i>	26
<i>Parsing a table from CSV or JSON data</i>	29
Performing operations on a table.....	32
<i>Creating calculated or formatted values</i>	32
<i>Sorting values alphabetically or numerically</i>	33
<i>Filtering rows from a table</i>	33
<i>Grouping rows</i>	33
<i>Joining tables</i>	34
<i>Breaking a table into pages</i>	34
<i>Rolling up date and time values</i>	35
<i>Selecting certain rows</i>	35
<i>Transposing a table</i>	35
Creating a table in real time	36
Getting a string that aggregates column values	36
Viewing the contents of nested tables	36
Unbinding and saving table data	37
Dataflow symbols and dataflow repeaters	38
Creating a dataflow symbol	39
<i>Creating a new dataflow symbol using block conversion</i>	39
<i>Creating a new dataflow symbol using the Symbols pane</i>	41
Editing a dataflow symbol.....	42
<i>Entering symbol editing mode</i>	43
<i>Editing a symbol dataflow model</i>	44
<i>Adding symbol parameters</i>	45
<i>Adding a dataflow symbol instance</i>	48
<i>Editing symbol instance properties</i>	49
Creating a dataflow repeater.....	49
Dataflow Symbol and Repeater tutorials.....	50
<i>Tutorial: Multiplication dataflow symbol</i>	50
<i>Tutorial: Multiplication dataflow repeater</i>	55
Dataflow blocks	58
Variables Blocks	59
<i>String</i>	59
<i>Number</i>	59
<i>Boolean</i>	60
<i>Table</i>	60
Data Services Blocks.....	61
<i>List Node</i>	61

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Conventions

<i>Get Default Parameters</i>	62
<i>String Loader</i>	63
<i>String Uploader</i>	65
<i>Get Children</i>	66
<i>Get Node</i>	67
<i>Multi-Histories</i>	69
<i>Zip Parser</i>	72
<i>Load Value</i>	73
<i>Load History</i>	74
<i>Invoke Action</i>	77
Logic Blocks.....	79
<i>And</i>	79
<i>Or</i>	79
<i>Not</i>	79
<i>If</i>	80
<i>Case</i>	81
<i>Hub</i>	83
<i>Event Gate</i>	84
<i>Script</i>	86
<i>Trace</i>	87
<i>Stop Watch</i>	87
<i>Delay</i>	89
<i>State</i>	89
<i>Catch Error</i>	91
<i>Make Object</i>	91
<i>Repeater</i>	92
String Operations Blocks.....	94
<i>Concatenate</i>	94
<i>Left</i>	95
<i>Length</i>	96
<i>Right</i>	96
<i>Substring</i>	97
<i>Replace</i>	98
<i>Trim</i>	99
<i>Split</i>	100
Math Operations Blocks.....	101
<i>Add</i>	101
<i>Divide</i>	101
<i>Factorial</i>	102
<i>Logarithm</i>	103

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Conventions

<i>Modulo</i>	104
<i>Multiply</i>	104
<i>Power</i>	105
<i>Root</i>	106
<i>Scale</i>	106
<i>Subtract</i>	107
Number Formatting Blocks	108
<i>Absolute</i>	108
<i>Bound</i>	109
<i>Round</i>	110
<i>Round Up</i>	111
<i>Round Down</i>	111
<i>Format Number</i>	112
<i>Parse Number</i>	113
Statistical Functions Blocks	114
<i>Average</i>	114
<i>Maximum</i>	115
<i>Median</i>	116
<i>Minimum</i>	117
<i>Mode</i>	118
<i>Standard Deviation</i>	119
<i>Variance</i>	120
Trigonometric Functions Blocks	121
<i>Arc Cosine</i>	121
<i>Arc Sine</i>	122
<i>Arc Tangent</i>	122
<i>Cosine</i>	123
<i>Cotangent</i>	123
<i>Degree</i>	124
<i>Radian</i>	124
<i>Sine</i>	125
<i>Tangent</i>	125
Table Operations Blocks	126
<i>CSV Parser</i>	126
<i>CSV Writer</i>	127
<i>JSON Parser</i>	128
<i>Column Mapping</i>	130
<i>Sort</i>	132
<i>Filter</i>	133
<i>Group By</i>	135

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Conventions

<i>Join</i>	137
<i>Page</i>	139
<i>Rollup</i>	141
<i>Aggregation</i>	143
<i>Select Rows</i>	144
<i>Select Columns</i>	145
<i>Get Columns</i>	146
<i>Table Row Cells</i>	147
<i>Add Row</i>	148
<i>Remove Rows</i>	149
<i>Edit Rows</i>	150
<i>Transpose</i>	152
<i>Realtime Recorder</i>	153
<i>Series Realtime Recorder</i>	154
Date Time Operations Blocks	155
<i>Date Time</i>	155
<i>Date Format</i>	157
<i>Parse Date Time</i>	158
<i>Date Math</i>	160
<i>Date Range</i>	161
Constants Blocks	163
<i>Pi</i>	163
<i>E</i>	164
<i>SQRT2</i>	164
<i>LN2</i>	165
<i>LN10</i>	165
<i>LOG2E</i>	166
<i>LOG10E</i>	166
Appendixes	167
Appendix 1: Dataflow Editor Script.....	167
Appendix 2: Dataflow Editor Syntax	167
Appendix 3: Scripts for debugging	170
Appendix 4: Reusing script.....	170
Appendix 5: Operators, Formats, and Functions	171
<i>Operators</i>	171
<i>String Usage and Functions</i>	173
<i>Number Formats and Functions</i>	176
<i>Date and Time Formats and Functions</i>	178
<i>Date and Time Functions</i>	182

**Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide**

Conventions

<i>List Functions</i>	184
<i>JSON Functions</i>	185
<i>XML Functions</i>	185
<i>ITEM 2\</item\>\</items\>'</i>	186
<i>);</i>	186
<i>var items = xml.queryAll('item');</i>	186
<i>Table Functions</i>	187
<i>tableClear(@parent.table);</i>	188
<i>Use of the dataflow/libs path</i>	189
Obtaining Documentation and Submitting a Service Request	190

Conventions

This document uses the following conventions.







Convention	Indication
bold font	Menu options, menu and submenu names, window titles, tab names, button names, and icon names appear in bold font .
<i>italic font</i>	New or emphasized terms are in <i>italic font</i> .
string	A string is a nonquoted set of characters. If quotation marks are included around the string, the string includes the quotation marks.
<code>courier font</code>	Filenames, directory names, folder names, code, script, and text that you type appear in <code>courier font</code> .
< >	Nonprinting characters such as passwords are in angle brackets.

Note: Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Caution: Means *reader be careful*. In this situation, you might perform an action that could result in equipment damage or loss of data.

User Interface Icons

This document refers to the following user interface icons.

Icon	Name
	Dataflow icon
	Edit in Window icon
	History icon
	Imported Layer icon
	New icon
	Remove icon

Dataflow Editor overview

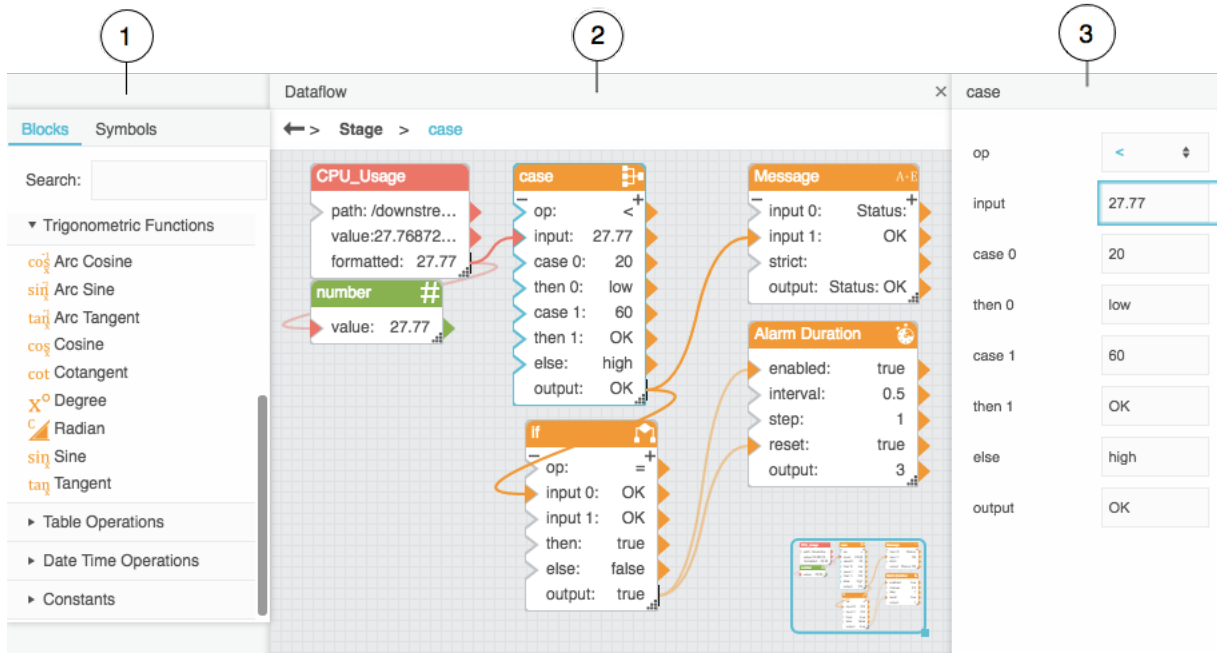
This overview explains the basic operations that the Dataflow Editor provides.

Structure of the Dataflow Editor

The Dataflow Editor is a visual programming environment. You use the Dataflow Editor to create *dataflow models*. A dataflow model is composed of *blocks* that do assigned tasks when requirements are met. A dataflow model is not a set of commands that are executed in sequence.

In the leftmost portion of the Dataflow Editor (1), the *Blocks pane* contains blocks that you can add to a dataflow model. In the center of the Dataflow Editor (2), the *Dataflow pane* displays a dataflow model. In the rightmost portion of the Dataflow Editor (3), the *Block Properties pane* shows properties of the currently selected block. Figure 1 shows the three portions of the Dataflow Editor.

Figure 1. Dataflow Editor



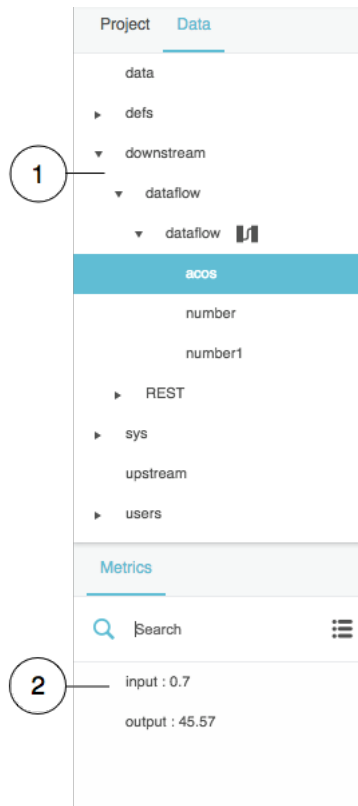
1	Blocks pane	2	Dataflow pane
3	Block properties pane		

To access your data, you use the Data pane and the Metrics pane. The Data pane contains the available data structure. When you select a node in the Data pane, the Metrics pane displays the metrics available at the selected node. Figure 2 shows the positions of the Data pane and Metrics pane.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow Editor overview

Figure 2. Data pane and Metrics pane



1	Data pane	2	Metrics pane
---	-----------	---	--------------

Each dataflow model belongs to a parent object. The parent of a dataflow model can be either of the following:

- Certain nodes in the Data pane
- Any dataflow block

Opening a dataflow model

Typically, only one dataflow model can be open at a time. The exception is dataflow symbols in symbol editing mode.

To open the dataflow model for a parent object, do one of the following:

- If the parent object is a data node, click the **Dataflow** icon next to the data node.
- If the parent object is a dataflow block, right-click the title bar of the dataflow block and choose **Dataflow**.

Figure 3 demonstrates how to open a dataflow model using the Data pane.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow Editor overview

Figure 3. Opening a dataflow model using the Data pane

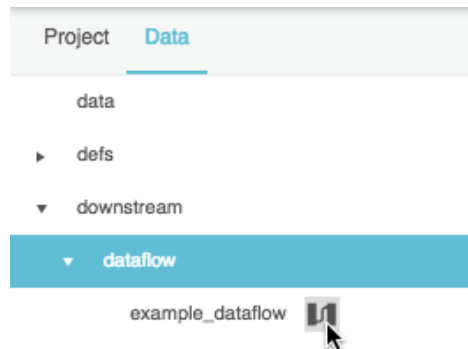
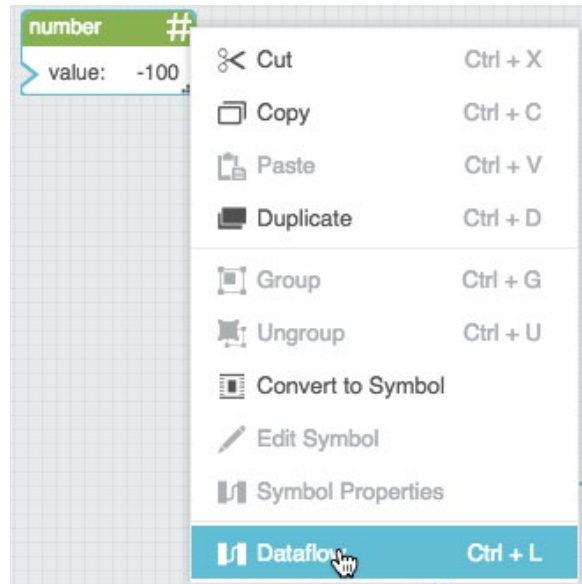


Figure 4 demonstrates how to open a dataflow model using the Dataflow pane.

Figure 4. Opening a dataflow model using the Dataflow pane



Dataflow blocks

When you create a dataflow model, you add blocks to the model and then edit the properties of the blocks.

This section describes how to add, select, rename, move, delete, and group dataflow blocks.

Dataflow blocks are represented by different colors, as described below:

- **Variable** blocks are green. These blocks store a value.
- **Data Services** blocks are red. These blocks access a data path and read or write information at that path or perform some other action.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow Editor overview

- **Operation** blocks are orange. These blocks receive an input value or values, perform an operation, and return the result.

Adding a block

You can add a block using one of these methods:

- The Blocks pane
- The Data pane
- The Metrics pane
- A binding

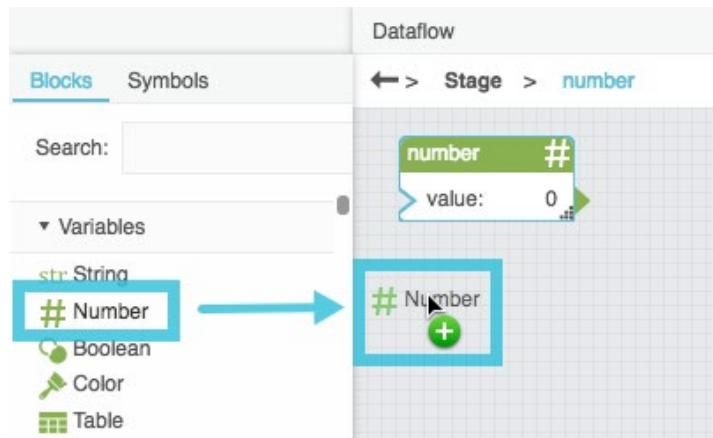
Adding a block using the Blocks pane

You can add most blocks using the Blocks pane by doing the following:

1. Find the block in the Blocks pane.
2. Do one of the following:
 - Drag the block and drop it on the Dataflow pane.
 - Click the block in the Blocks pane.

Figure 5 demonstrates how to drag a block from the Blocks pane.

Figure 5. Dragging a block from the Blocks pane



Note: You can use the Blocks pane's **Search** function to find blocks quickly.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow Editor overview

Adding a Get Children Block using the Data pane

As a shortcut, to add a Get Children block, drag the relevant node from the Data pane and drop the node on the Dataflow pane.

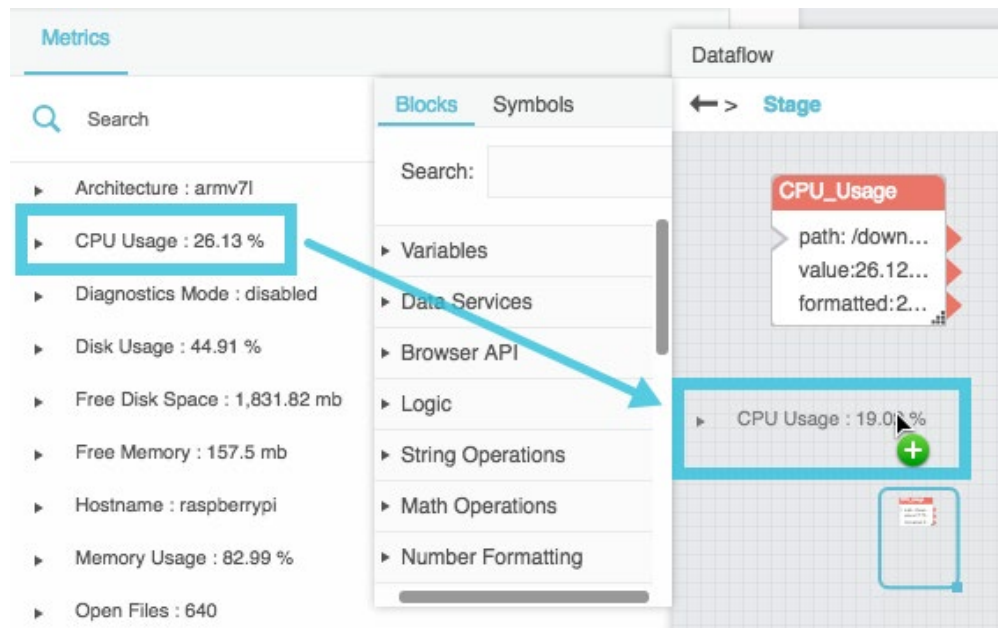
Adding a block using the Metrics pane

The following interactions add blocks using the Metrics pane:

- As a shortcut, to add a Load Value block, drag a metric from the Metrics pane and drop the metric on the Dataflow pane.
- As a shortcut, to add a Load History block, drag a **History** icon from the Metrics pane and drop the icon on the Dataflow pane.
- To add an Invoke Action block, right-click a node to see a list of available actions and then drag an action to the Dataflow pane. See Invoke Action. This block cannot be added using the Blocks pane.

Figure 6 demonstrates how to add blocks using the Metrics pane.

Figure 6. Dragging a block from the Metrics pane



Selecting a block

When you select a block, you can move the block and see its properties in the Block Properties pane.

Selected blocks have a blue outline in the Dataflow pane.

The following interactions affect block selection:

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

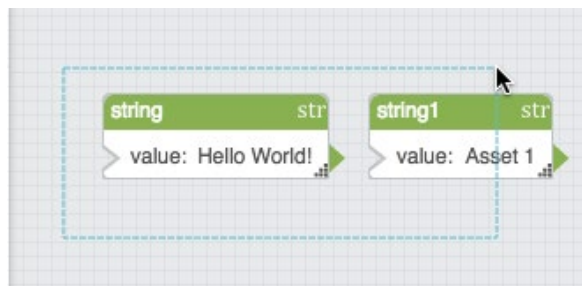
Dataflow Editor overview

- To select one block, click the block title bar. Figure 7 demonstrates selection of one block.
- To select multiple blocks, click an empty area in the **Dataflow** pane and drag to create a selection rectangle. Figure 8 demonstrates how to drag to select blocks.
- To add a block to the selection or remove a block from the selection, hold **Shift on** (Mac) or **Ctrl on** (Windows) and click the block's title bar.

Figure 7. Selecting a single block



Figure 8. Selecting multiple blocks



Renaming a block

You can give blocks meaningful names to keep your dataflow easy to read.

You can change the label only or you can change both the label and Dataflow Editor syntax name of the block. You might want to change the Dataflow Editor syntax name if you use script heavily.

Caution: Changing the Dataflow Editor syntax name of a block breaks any bindings or scripts that already use that block. You must also change the name in these bindings or scripts.

To rename a block:

1. Double-click the block title bar.
2. Edit the block name.
3. Press **Enter** or **Return** to change the label only or press **Ctrl + Enter** to also change the name of the block in the page code.

Figure 9 demonstrates how to rename a block.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow Editor overview

Figure 9. Renaming a block



Moving a block

You can move blocks to change their visual positions in the **Dataflow** pane.

- To move a block, click the block title bar and drag. This step moves all blocks that are selected.

Note: The **Snapping** feature helps you precisely place blocks. When Snapping is enabled, guidelines appear when a block that you are dragging is aligned to other blocks and it is easier to place blocks in aligned positions. To enable or disable Snapping, choose **View > Snapping > Snapping in Dataflow**.

Deleting blocks

To delete blocks:

1. Select one or more blocks.
2. Press the **Delete** or **Backspace** key.

Grouping blocks

When you group dataflow blocks, the blocks are replaced by a single orange block that represents the group and the grouped blocks become part of this new block's dataflow model. You can right-click the new orange block and choose **Dataflow** to see the blocks inside the group.

To group dataflow blocks:

1. Select one or more blocks.
2. Right-click a selected block and choose **Group**.

Ungrouping blocks

To ungroup dataflow blocks:

1. Select the block that represents the group.
2. Right-click the selected block and choose **Ungroup**.

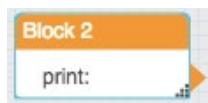
Block properties

Blocks can have two types of properties: input/output properties and output properties. An input/output property can take an input value and give an output value. Figure 10 shows an input/output property. An output property can give an output value but cannot take an input value. Figure 11 shows an output property.

Figure 10. Input/Output property



Figure 11. Output property



Adding and removing Input/Output properties

For certain blocks, you can add and remove input/output properties. To add properties, click the plus sign (+) in the block. To remove properties, click the minus sign (-). There is no limit to the number of input properties for such a block.

Figure 12 shows a block that allows properties to be added and removed.

Figure 12. Block with a configurable number of properties



Pinning and unpinning properties

All of a block's properties are shown in the Block Properties pane. Properties are visible in the Dataflow pane only if they are *pinned*. Certain properties are pinned by default. You can pin and unpin properties.

To pin a property of a block:

1. In the **Block Properties** pane, hover over the property until a blue dot appears.
2. Click the blue dot and check the **Pinned** check box.

To unpin a property of a block, uncheck the **Pinned** check box.

Figure 13 shows an unpinned property. Figure 14 shows a pinned property.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow Editor overview

Figure 13. Unpinned property

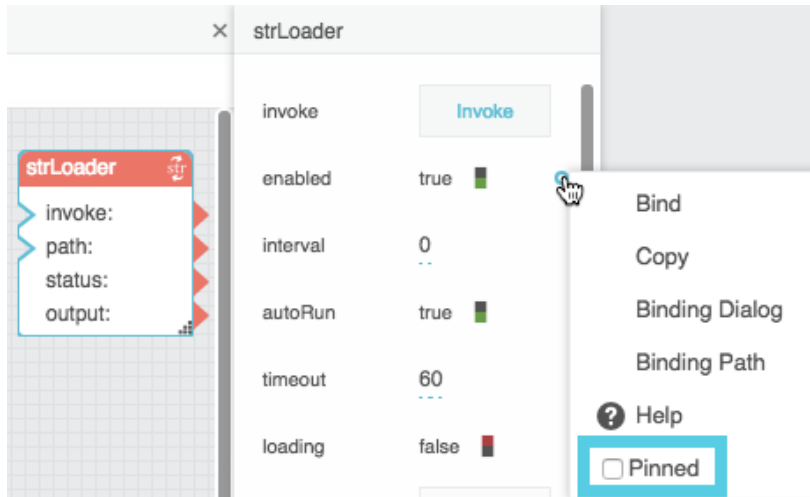
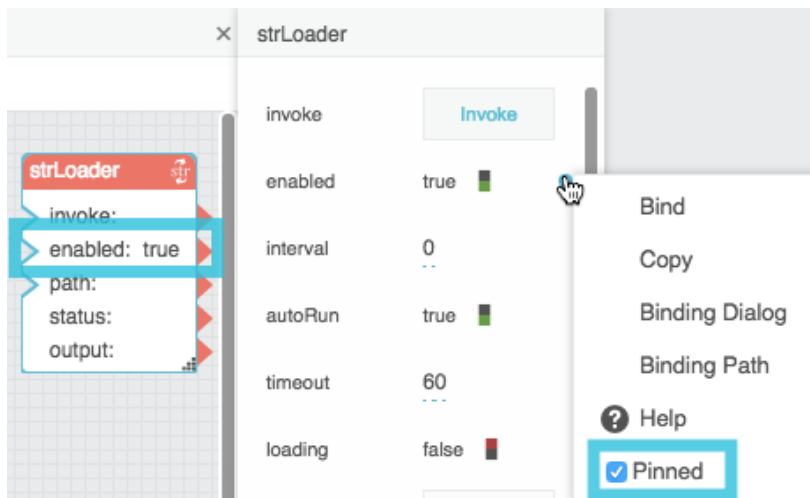


Figure 14. Pinned property



Manually setting property values

You can manually set the value of an input/output block property using the Block Properties pane.

To set property values:

1. Select the block.
2. In the **Block Properties** pane, edit the value.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow Editor overview

Figure 15. Setting a property value



Viewing Help for a property

You can view in-product Help for properties.

To open Help for a property:

1. Hover over the property until a blue dot appears.
2. Click the blue dot and select **Help**.

Bindings

You can cause a *source* block property to determine a *target* block property. Such connections are called *bindings* and are represented with wires.

Using arrowheads to create bindings to block properties

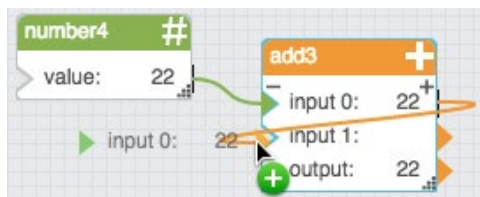
All pinned properties have an arrowhead on the right that you can drag to create bindings. Only input/output properties have a cutout triangle on the left that can take an input.

To use an arrowhead to create a binding:

1. Click and drag the arrowhead of the source property.
2. Drop the arrowhead on the cutout triangle of the target property.

Figure 16 demonstrates how to create a binding between two properties of the same block. Figure 17 demonstrates how to create a binding between two blocks. Figure 18 shows some completed bindings.

Figure 16. Creating a binding between properties of the same block



Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow Editor overview

Figure 17. Creating a binding between blocks

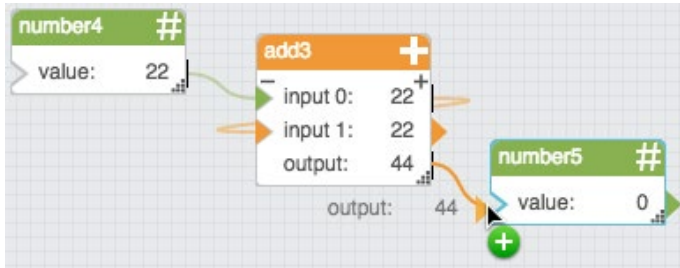


Figure 18. Completed bindings

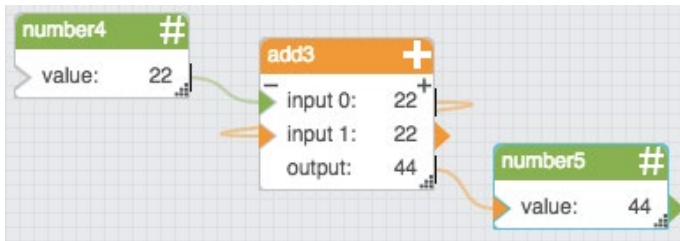
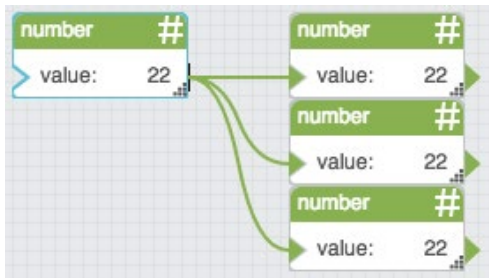


Figure 19 shows one source property that has been bound to multiple target properties.

Figure 19. Source property with multiple target properties



Creating bindings to and from the Block Properties pane

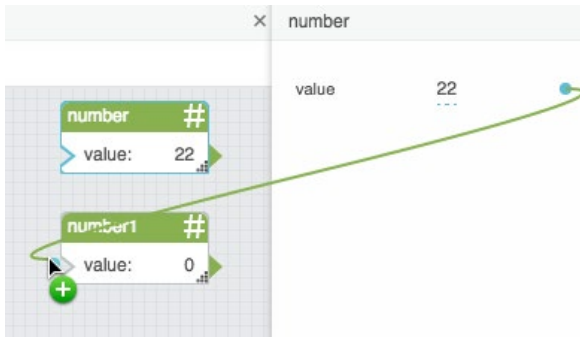
You can also create bindings using the Block Properties pane.

Figure 20 shows a how to create a binding from the Block Properties pane to the Dataflow pane.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow Editor overview

Figure 20. Creating a binding from the Block Properties pane to the Dataflow pane



Event and trigger properties

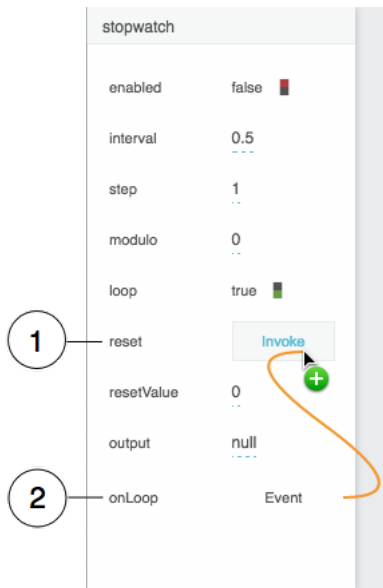
Some properties are *events*. Event properties fire automatically when something happens, such as the completion of a data process.

Other properties are *triggers*. Trigger properties cause things to occur, such as the beginning of a data process.

You can bind an event property to a trigger property to cause a chain of events. When the source event occurs, the target trigger causes the next event to occur. For example, you can cause a stopwatch to begin after a string is loaded.

Figure 21 shows an example of a trigger property and an event property.

Figure 21. Examples of trigger and event properties



1	Trigger property	2	Event property
----------	------------------	----------	----------------

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow Editor overview

Finding the valid string to bind to an enum property

You can bind a string property to an enum property. Only one specific, case-sensitive string is valid for each enum value.

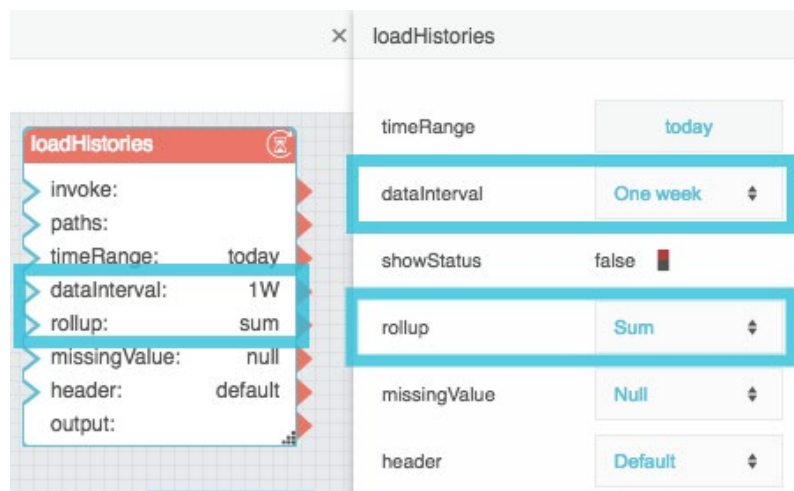
To see the valid string:

1. Make sure the enum property is pinned. See Pinning and unpinning properties.
2. Choose a value for the enum property.

The valid string appears in the visual block in the Dataflow pane. The valid string might be different from the value in the drop-down list in the Block Properties pane.

Figure 22 shows an example of how to find valid strings for enum values. In this example, for the **dataInterval** property, **1W** is a valid string. For the **rollup** property, **sum** is a valid string.

Figure 22. Valid strings for string-to-enum binding

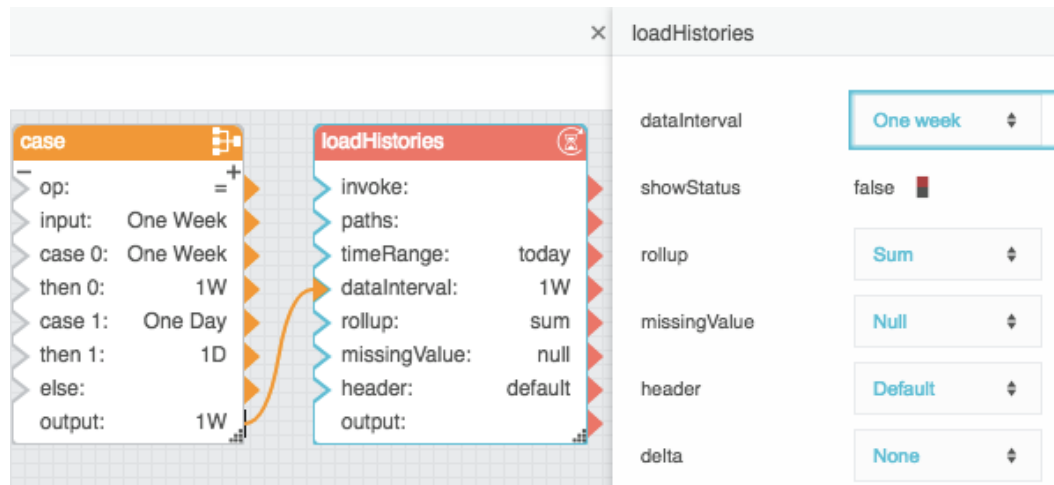


In Figure 23, the list option **One Week** corresponds to the Case block output **1W**. This Case block output is bound to the enum property.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow Editor overview

Figure 23. Example of a string-to-enum binding



Using Dataflow Editor syntax to review or edit a binding

To review the source or targets of a property's bindings:

1. Hover over the property until a blue dot appears.
2. Click the blue dot and select **Binding Path** to review the source or **Binding Targets** to review the targets.

A pop-up appears. The pop-up lists the path or targets in Dataflow Editor syntax.

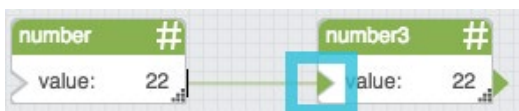
Using the arrowhead to delete a binding

Deleting a binding does not typically change the value of the property that was the target.

To delete a binding to a block:

- Double-click the filled-in triangle of the target property, as shown in Figure 24.

Figure 24. How to Delete a Binding



The binding and the wire that depicted it are deleted.

Using the Block Properties pane to delete a binding

Deleting a binding does not typically change the value of the property that was the target.

To delete a binding:

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow Editor overview

1. Hover over the property in the Block Properties pane until a blue dot appears.
2. Click the blue dot and choose **Unbind**.

Deleting and resetting a binding

To delete a binding and also reset the property to its default value:

1. Hover over the property in the Block Properties pane until a blue dot appears.
2. Click the blue dot and choose **Reset**.

Copying and pasting a path or property value

To copy a path or value from one target property and re-use that path or value for another target property:

1. Hover over the property in the Block Properties pane until a blue dot appears.
2. Click the blue dot and choose **Copy**.
3. Navigate to the property that you want to receive the value.
4. Click the blue dot and choose **Paste**.

The exact binding path string or the property value is pasted. See also Appendix 2: Dataflow Editor Syntax.

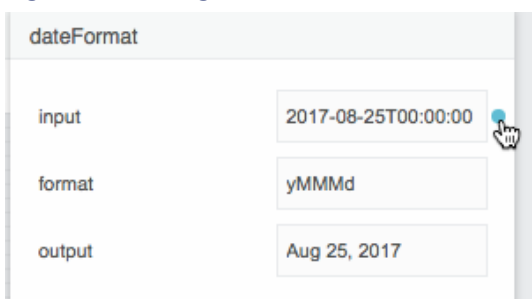
Using the small Binding popup window

Sometimes you might want to create a binding from an unpinned property of one block to an unpinned property of another block. To do this, you can use a small Binding popup window.

To use the small Binding popup window:

1. In the Block Properties pane, hover over either the source property or the target property until a blue dot appears. See Figure 25.

Figure 25. Finding the Blue Dot menu



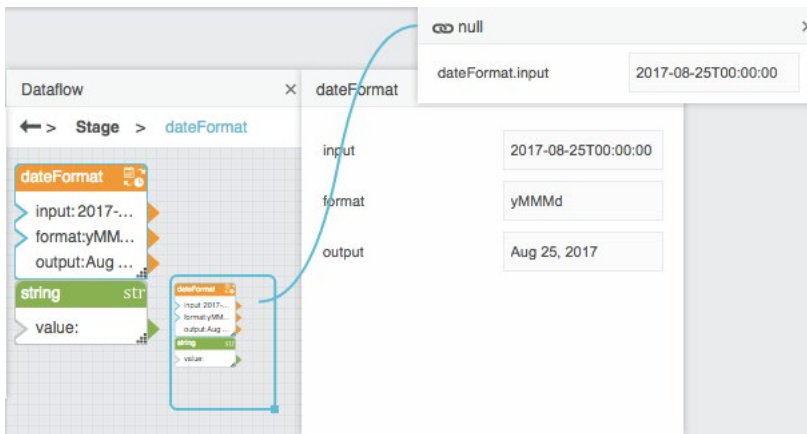
2. Double-click the blue dot or click the blue dot and choose **Bind**.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow Editor overview

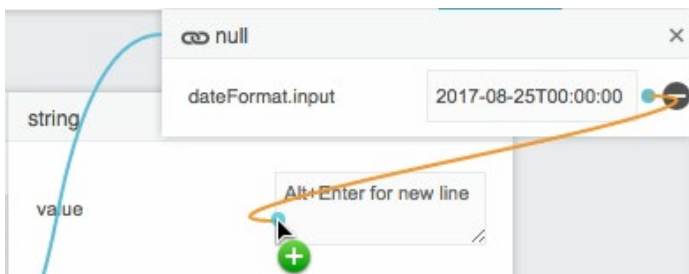
A small Binding popup window appears, as shown in Figure 26.

Figure 26. Small Binding popup window



3. In the Block Properties pane, navigate to the other property.
4. Click the blue dot of the source property and drag it to the target property, as shown in Figure 27.

Figure 27. Creating a binding using the small Binding popup window



The binding is formed.

Notes:

- The small Binding popup window can hold several properties at once.
- To remove a property from the Binding popup window, hover over the property and click the **Remove** icon.
- If you hover over any property and a blue dot appears, this indicates that the property can be used in a binding.

Tables

One purpose of the Dataflow Editor is to work with tables. This section covers:

- How to load a table
- How to perform operations on a table
- How to create a table in real time
- How to get a string that aggregates column values
- How to view the contents of nested tables
- How to unbind and save table data

Note: The Dataflow Editor does not support duplicate column names within a table. In particular, the Column Mapping block and the Filter block do not support duplicate column names.

Loading a table

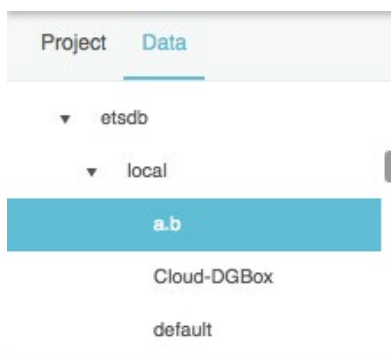
To load a table, first open the dataflow model, then follow the relevant procedure below.

Loading a table from a metric history

To load a table from a metric history:

1. In the **Data** pane, select the relevant node, as shown in Figure 28

Figure 28. Selecting a Data Pane node

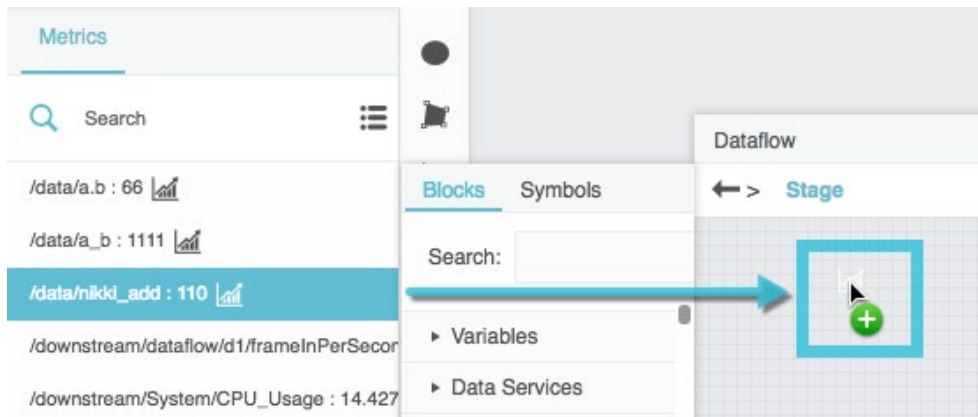


2. In the **Metrics** pane, find the relevant data metric and drag its **History** icon to the **Dataflow** pane, as shown in Figure 29.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

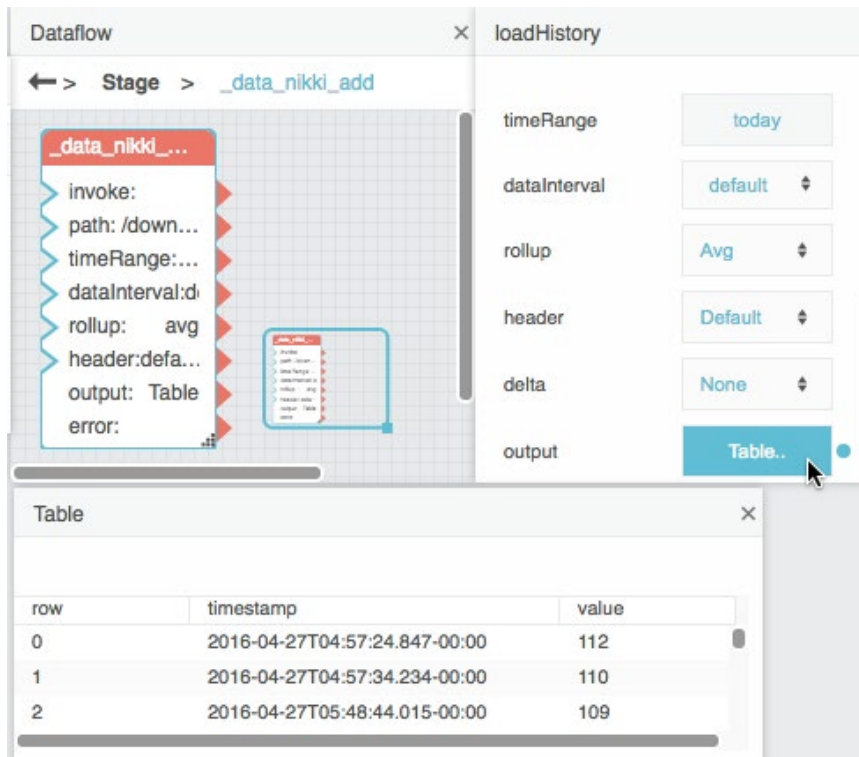
Tables

Figure 29. Adding a Load History block



- To open the table, click the **Load History** dataflow block and, in the **Block Properties** pane, click the value of the **output** property, as shown in Figure 30.

Figure 30. Loaded History



Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Tables

Loading a table from multiple metric histories

To load one table from multiple data metric histories:

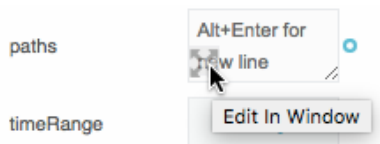
1. In the **Dataflow** pane, expand **Data Services** and drag a **Multi-Histories** block to the **Dataflow** pane, as shown in Figure 31.

Figure 31. Adding a Multi-Histories block



2. Select the Multi-Histories block.
3. In the **Block Properties** pane, click the **Edit in Window** icon in the **paths** field, as shown in Figure 32.

Figure 32. Editing in the Window Icon



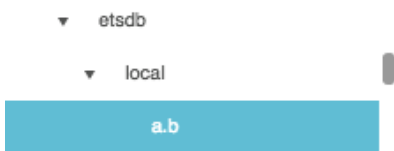
A popup window appears.

4. In the **Data** pane, select the relevant data source, as shown in Figure 33.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

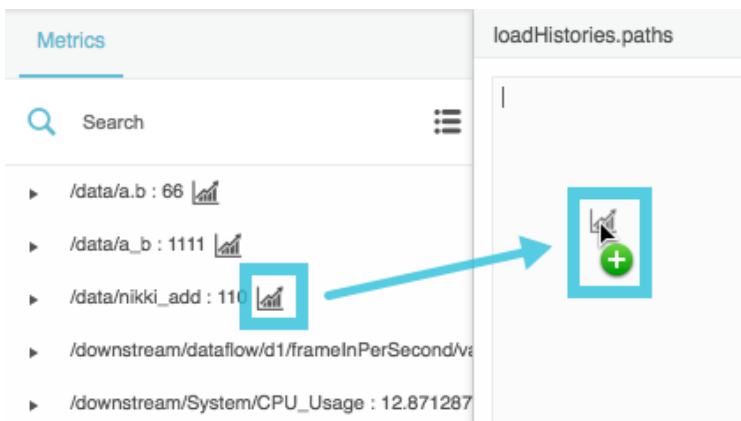
Tables

Figure 33. Selecting a Data pane node



5. In the **Metrics** pane, select a relevant data metric and drag its **History** icon to the popup window, as shown in Figure 34.

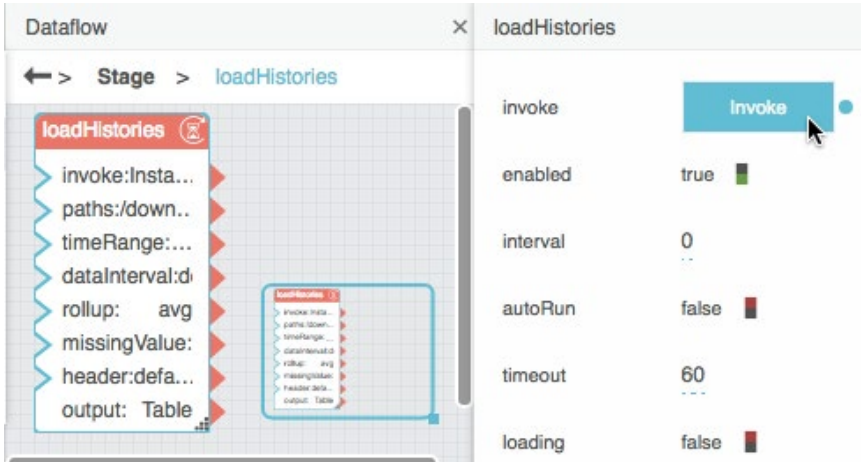
Figure 34. Adding a history to the Multi-Histories block



6. After the path, press **Enter** to insert a line break.
7. Repeat Steps 4 to 6 until all of the relevant data metrics are loaded and then click **Apply** or **OK**.
8. With the **Multi-Histories** block selected, click **Invoke**, as shown in Figure 35.

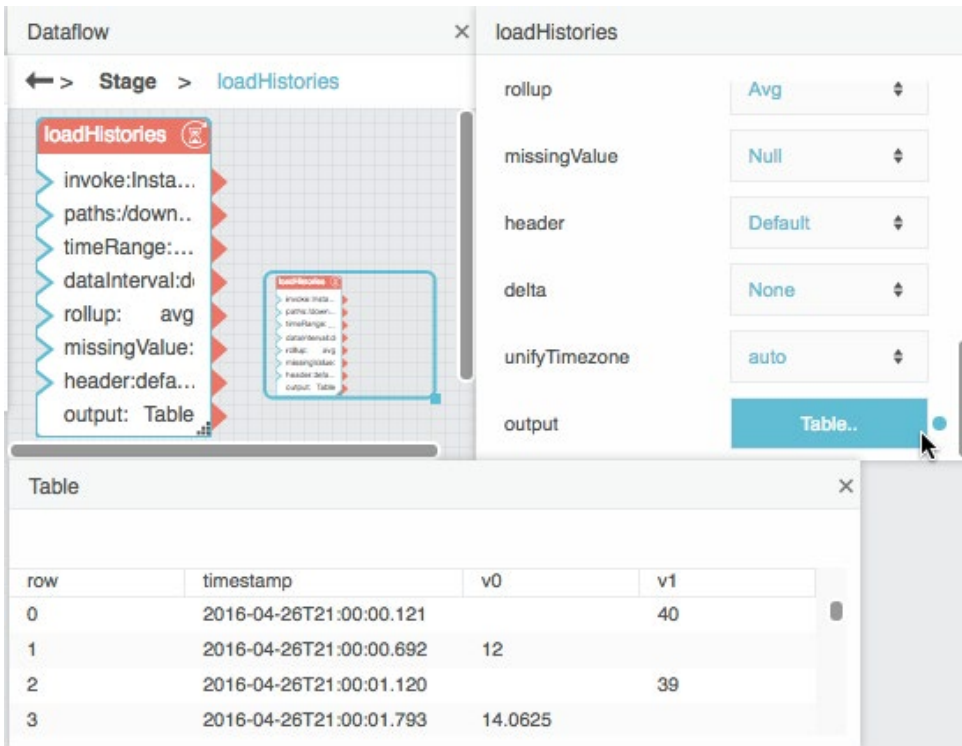
Tables

Figure 35. Invoking a Load Histories block



9. With the **Multi-Histories** block selected, click the value of the **output** property to open the table, as shown in Figure 36.

Figure 36. Multiple loaded histories



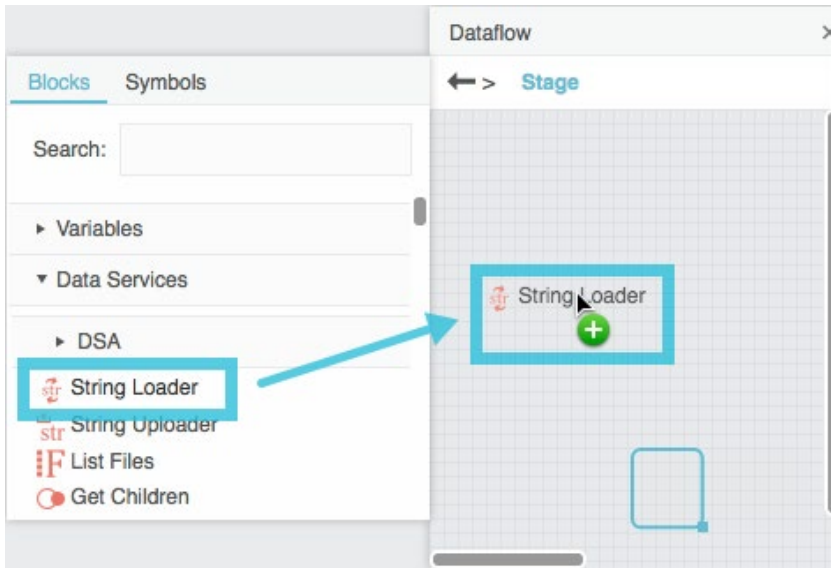
Tables

Parsing a table from CSV or JSON data

To load a CSV or JSON string and then parse it into a table:

1. In the **Blocks** pane, expand **Data Services** and drag a String Loader block onto the **Dataflow** pane, as shown in Figure 37.

Figure 37. Adding a String Loader block

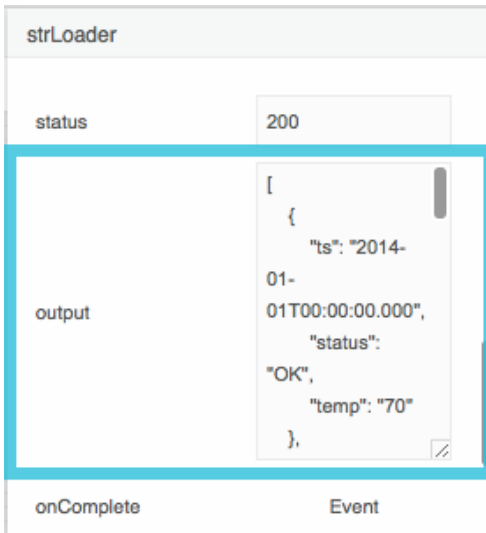


2. With the **String Loader** block selected, enter the path of the CSV or JSON file in the path property and press **Enter** or **Return**.

The CSV or JSON string appears as the **output** property, as shown in Figure 38.

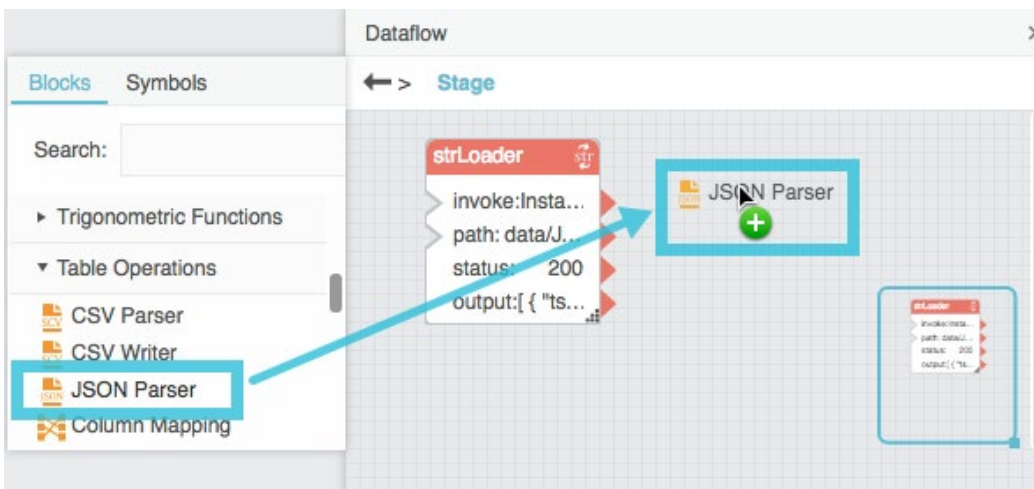
Tables

Figure 38. Loaded String as output property



- Expand **Table Operations** and drag a CSV Parser or JSON Parser block onto the **Dataflow** pane, as shown in Figure 39.

Figure 39. Adding a JSON Parser block



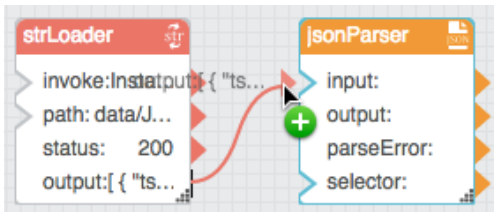
- Bind the **output** property of the String Loader block to the **input** property of the CSV Parser or JSON Parser block, as shown in Figure 40.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

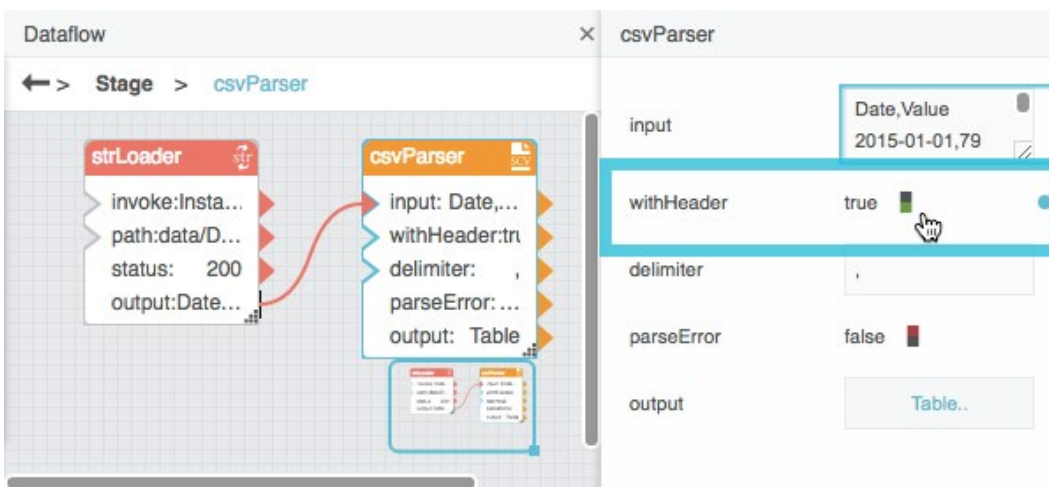
Tables

Figure 40. Binding a String Loader to the JSON parser



- If your file is a CSV file that includes headers, choose the **CSV Parser** block and set the **withHeader** property to **true**, as shown in Figure 41.

Figure 41. Setting withHeader to TRUE



- Choose the **CSV Parser** or **JSON Parser** block and then click the value of the **output** property to open the table, as shown in Figure 42.

Figure 42. Parsed CSV file

The screenshot shows the Dataflow Editor interface. On the left, a dataflow graph displays a `strLoader` stage connected to a `csvParser` stage. The `csvParser` stage configuration is shown on the right, with the following properties:

- `input`: Date, Value (with a preview of 2015-01-01,79)
- `withHeader`: true
- `delimiter`: ,
- `parseError`: false
- `output`: Table..

Below the configuration, a table view displays the parsed data:

row	Date	Value
0	2015-01-01	79
1	2015-01-02	88
2	2015-01-03	71
3	2015-01-04	53

Performing operations on a table

You can use dataflow to perform an operation on an input table and return the result as an output table.

Creating calculated or formatted values

To create a new column that contains calculated or formatted values:

1. If the source data comes from multiple tables, use one or more Join blocks to create a unified table.
2. Add a **Column Mapping** block to the dataflow model.
3. Bind the input table to the **input** property of the **Column Mapping** block.
4. Specify the new column name, as the **name 0** property.
5. Specify the calculated or formatted value, as the **from 0** property. Use JavaScript notation. For JavaScript examples, see Column Mapping and Appendix 5: Operators, Formats, and Functions.

The output table appears as the **output** property of the Column Mapping block.

6. To create additional columns, click the plus sign (+) to add **name n** and **from n** values to the block. Then, repeat Steps 4 and 5.

See also Column Mapping.

Tables

Sorting values alphabetically or numerically

To sort table rows in alphabetical or numerical order:

1. Add a **Sort** block to the dataflow model.
2. Bind the input table to the **input** property of the **Sort** block.
3. With the **Sort** block selected, specify the following in the **Block Properties** pane:
 - The column name from the input table, as the **column** property
 - The sort method (alphabetical or numerical), as the **method** property
 - The sort order (ascending or descending), as the **order** property

The output table appears as the **output** property of the **Sort** block.

Note: You can also use a **Select Rows** block to set the row order manually.

See also Sort.

Filtering rows from a table

To filter rows from your data:

1. Add a Filter block to the dataflow model.
2. Bind the input table to the **input** property of the Filter block.
3. Write a condition in JavaScript notation, and specify this condition as the **condition** property. For condition examples, see Filter and Appendix 5: Operators, Formats, and Functions.

The output table appears as the **output** property of the Filter block.

Notes:

- You can also use a **Select Rows** block to manually exclude rows.
- To restrict data to a date and time range, you can also use the **timeRange** property of a Load History or Multi-Histories block.

See also Filter.

Grouping rows

To ensure that only one row appears in the table for each unique value:

1. Add a **Group By** block to the dataflow model.
2. Bind the input table to the **input** property of the **Group By** block.

Tables

3. With the **Group By** block selected, specify the following in the **Block Properties** pane:
 - The column on which grouping is determined, as the **baseColumn** property.
 - Other columns to include from the input table, as **column n** properties.
 - The grouping method for each column, as **method n** properties. For descriptions, see Group By.
 - The name of each column in the output table, as **outColumn n** properties.

The output table appears as the **output** property of the Group By block.

Note: To combine rows based on date and time intervals instead of duplicate values, you can either use a **Rollup** block or use the **dataInterval** and **rollup** properties of a **Load History** or **Multi-Histories** block.

See also Group By.

Joining tables

To perform a join operation that combines two input tables:

1. Add a **Join** block to the dataflow model.
2. Bind the left input table to the **input1** property of the Join block.
3. Bind the right input table to the **input2** property of the Join block.
4. With the **Join** block selected, specify the following in the **Block Properties** pane:
 - The key column in the left input table, as the **column1** property.
 - The key column in the right input table, as the **column2** property.
 - The join method to use, as the **join** property. For descriptions, see Join.

The output table appears as the **output** property of the Join block.

See also Join.

Breaking a table into pages

To break up your data into sequential sections:

1. Add a **Page** block to the dataflow model.
2. Bind the input table to the **input** property of the **Page** block.
3. With the **Page** block selected, specify the following in the **Block Properties** pane:
 - The number of rows per page, as the **pageSize** property
 - The row index on which the current page begins, as the **start** property

Tables

The table portion appears as the **output** property of the Page block.

See also Page.

Rolling up date and time values

To ensure that only one row appears for each date and time interval:

1. Add a **Rollup** block to the dataflow model.
2. Bind the input table to the **input** property of the **Rollup** block.
3. With the **Rollup** block selected, specify the following in the **Block Properties** pane:
 - The length of the **interval**.
 - The input table column that holds dates, as the **dateColumn** property.
 - The input table column that holds values, as the **valueColumn** property.
 - The type of rollup to use, as the **valueRollup** property. For descriptions, see Rollup.

The output table appears as the **output** property of the Rollup block.

Note: You can also use the **dataInterval** and **rollup** properties of a Load History or Multi-Histories block.

See also Rollup.

Selecting certain rows

To include only certain rows manually:

1. Add a **Select Rows** block to the dataflow model.
2. Bind the input table to the **input** property of the **Select Rows** block.
3. With the **Select Rows** block selected, specify which rows to include, as a list of comma-separated numbers.

The output table appears as the **output** property of the **Select Rows** block.

Note: You can also use a **Select Rows** block to manually set the row order.

See also Select Rows.

Transposing a table

To transpose an input table, so that the columns in the input table become the rows in the output table:

1. Add a **Transpose** block to the dataflow model.
2. Bind the input table to the **input** property of the **Transpose** block.

Tables

The output table appears as the **output** property of the **Transpose** block.

See also Transpose.

Creating a table in real time

To monitor changes to specified values and create a table that records values as they change:

1. Add a Realtime Recorder block to the dataflow model.
2. Bind the value that you want to monitor to the **value 0** property.
3. Specify the name of the value, as the **name 0** property.
4. To monitor additional values, click the plus sign (+) to add **name n** and **value n** values to the block. Then, repeat Steps 2 and 3.

The output table appears as the **output** property of the Realtime Recorder block. This table updates when the values change.

Note: Realtime Recorder table contents are saved only in the current session.

See also Realtime Recorder.

Getting a string that aggregates column values

To get a string that reflects the table records in a column:

1. Add an **Aggregation** block to the dataflow model.
2. Bind the input table to the **input** property of the **Aggregation** block.
3. With the **Aggregation** block selected, specify the following:
 - The column to aggregate, as the **column** property.
 - The method of aggregation, as the **method** property. For descriptions, see Aggregation.

The string appears as the **output** property of the Aggregation block.

See also Aggregation.

Viewing the contents of nested tables

Nested tables occur whenever a table stores other tables.

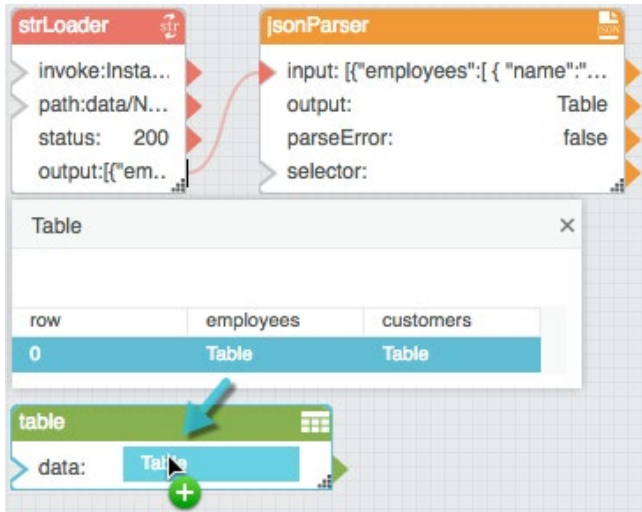
To view a table that is in another table:

1. Add a **Table** block to the dataflow model.

Tables

2. Drag the *cell* that contains a table and drop the cell on the **Table** block, as shown in Figure 43.

Figure 43. Binding a nested table to a table block



This creates a binding from the table in the cell to the Table block. To view the table data, you can click the value of the Table block's **data** property.

Unbinding and saving table data

Sometimes you might want to delete a binding to a table while also preserving the data that the table currently holds. For example, you might want to share a dataflow model with a colleague who does not have access to your data server.

To unbind a table and preserve its data:

1. Make sure that the data is bound to a **Table** block.
2. With the **Table** block selected, invoke the **save** property, as shown in Figure 44.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

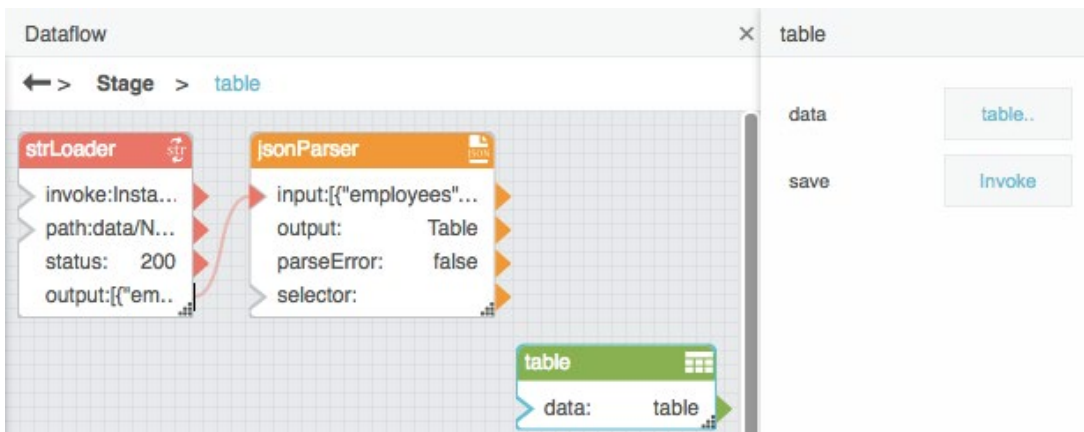
Dataflow symbols and dataflow repeaters

Figure 44. Unbinding and saving table data



The binding to the **data** property is destroyed and the data is saved in the Table block, as shown in Figure 45.

Figure 45. Unbound and saved table data



Dataflow symbols and dataflow repeaters

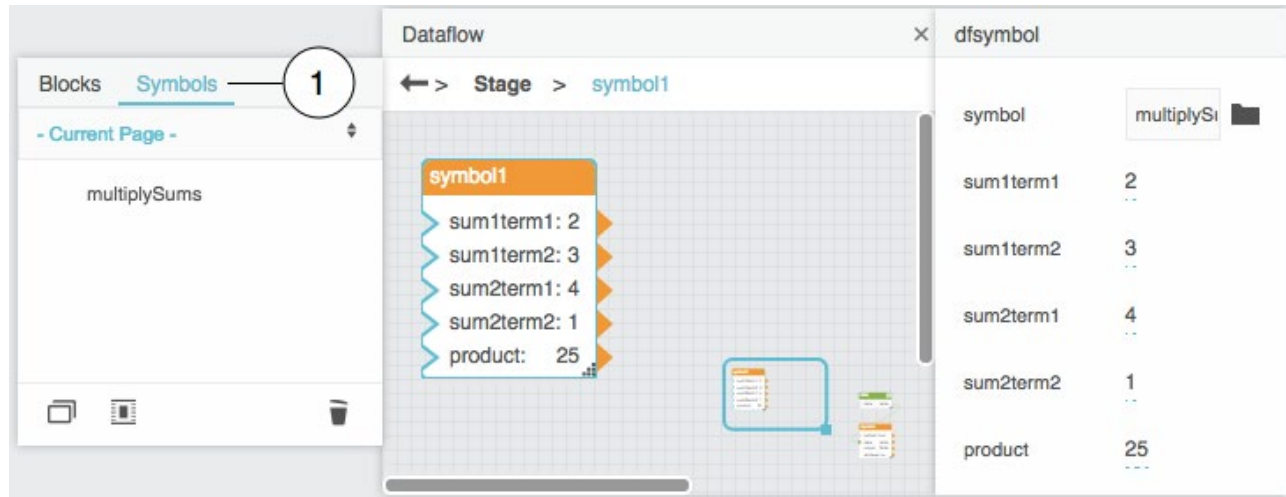
This section describes dataflow symbols and dataflow repeaters and includes basic operations and tutorials.

A *dataflow symbol* is a dataflow model without a parent object. You access dataflow symbols in the *Symbols pane* of the Dataflow Editor. The Symbols pane is located in a tab next to the Blocks pane. Figure 46 shows the location of the Symbols pane.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

Figure 46. Location of the Symbols pane



1	Symbols pane
---	--------------

After you create a dataflow symbol, you can add *instances* of that symbol to dataflow models. A symbol instance is a single dataflow block that represents the symbol. Instances of a symbol are identical to one another except where affected by symbol *parameters* that you create. If a symbol does not have parameters, all of its instances are identical.

You can edit a symbol or you can edit a symbol instance. Changes to a symbol affect all instances of the symbol. Changes to an instance affect only that instance. Only the following aspects of an instance can be edited:

- Position and size.
- Block label and Dataflow Editor syntax name.
- Which properties are pinned and unpinned.
- Parameter values.

A *dataflow repeater* is a Repeater block. A Repeater block takes a table and a dataflow symbol as input. Then, the Repeater block returns an output table that describes parameter values of the input symbol for each row of the input table.

Creating a dataflow symbol

You can create a new dataflow symbol in the following two ways.

Creating a new dataflow symbol using block conversion

To create a new dataflow symbol:

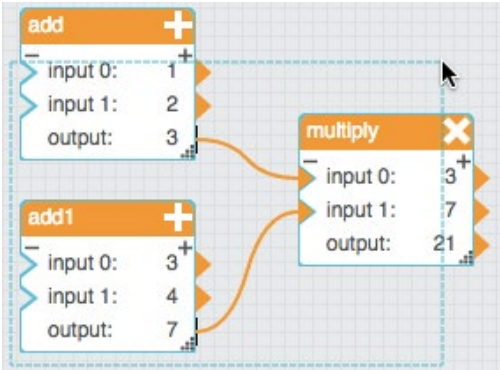
1. Add blocks to any dataflow model.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

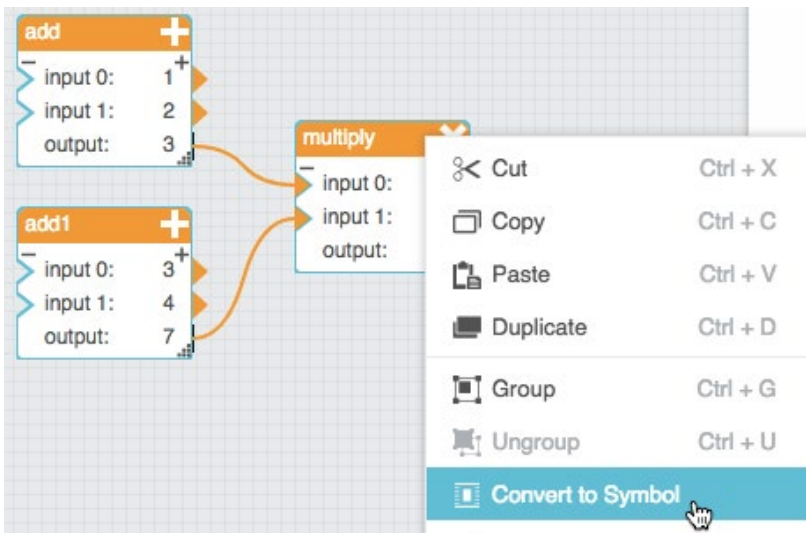
2. Choose the blocks to convert, as shown in Figure 47.

Figure 47. Selecting blocks



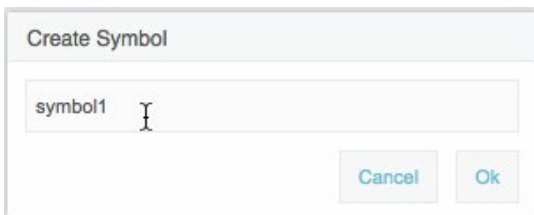
3. Right-click one of the selected blocks and choose **Convert to Symbol**, as shown in Figure 48.

Figure 48. Converting blocks to symbol



4. When prompted, enter a symbol name, as shown in Figure 49, and click **OK**.

Figure 49. Creating a symbol name

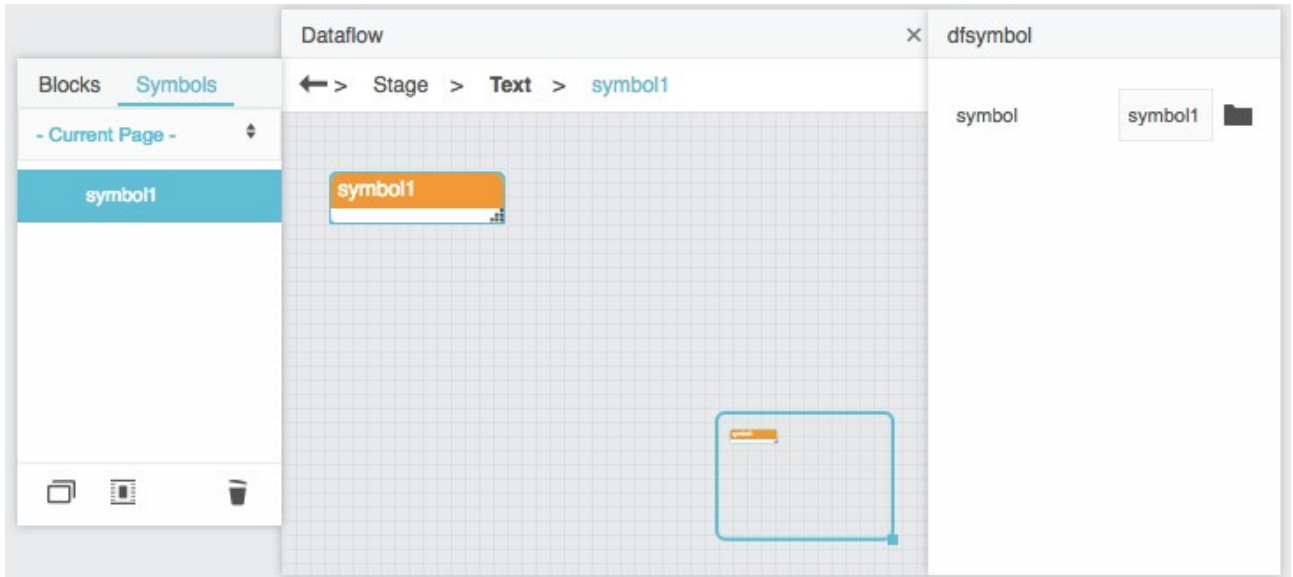


Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

In the current dataflow model, the selected blocks are replaced by a symbol instance. In the Symbols pane, the new symbol appears in the symbols list, as shown in Figure 50.

Figure 50. Result of block-to-symbol conversion



Creating a new dataflow symbol using the Symbols pane

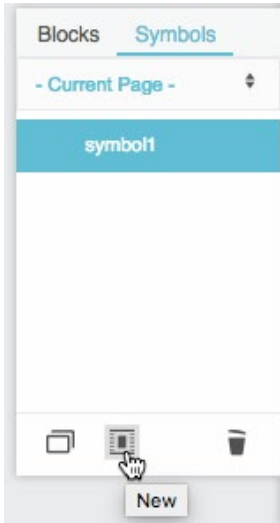
To create a new dataflow symbol and also enter symbol editing mode for that symbol:

1. Make sure the **Symbols** pane, not the Blocks pane, is selected.
2. Click the **New** icon, as shown in Figure 51.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

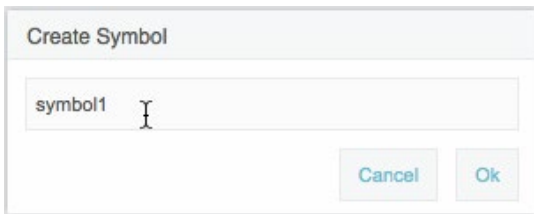
Dataflow symbols and dataflow repeaters

Figure 51. New icon in the Symbols pane



3. When prompted, enter a symbol name, as shown in Figure 52, and click **OK**.

Figure 52. Creating a symbol name



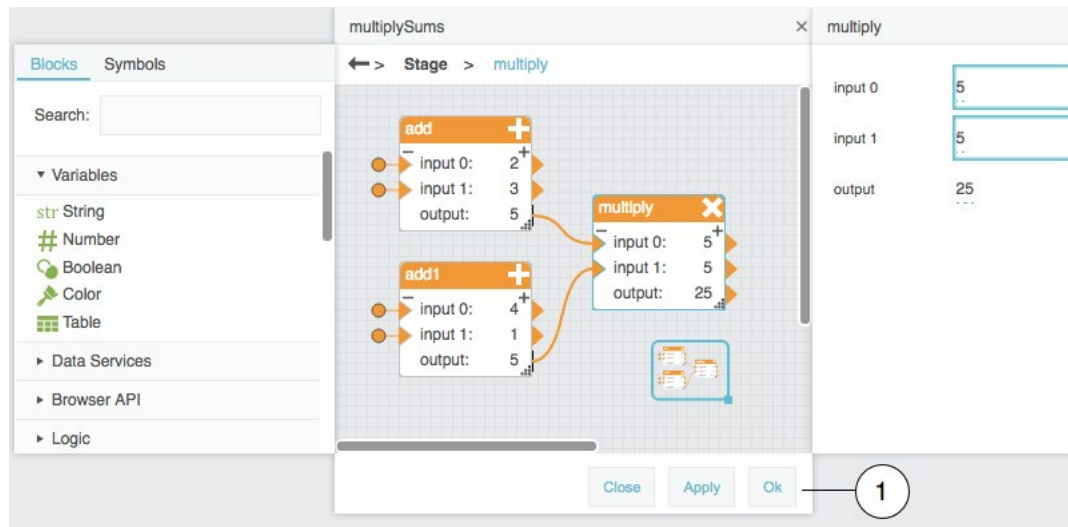
Editing a dataflow symbol

You edit a dataflow symbol in *symbol editing mode* for that symbol. When you are in symbol editing mode, the Dataflow Editor window contains the dataflow model for the symbol and three buttons let you close, apply, or save the current dataflow model. These buttons appear only in symbol editing mode. Figure 53 shows symbol editing mode.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

Figure 53. Symbol editing mode



1	Symbol editing mode buttons
---	-----------------------------

If you are in symbol editing mode for multiple symbols simultaneously, the Dataflow Editor window for each symbol appears in a separate popup window.

While in symbol editing mode, you can make two kinds of changes. The first kind of change affects the symbol dataflow model. The second kind of change exposes and manages symbol parameters.

Notes:

- You always must click **OK** to save changes. Changes to a dataflow symbol are not saved automatically when you leave symbol editing mode.
- While you are in symbol editing mode, you can click **Apply** to apply changes to all symbol instances temporarily. If you click **Apply** but do not click **OK**, your changes are lost when you exit symbol editing mode.

Entering symbol editing mode

The following interactions enter symbol editing mode for a dataflow symbol:

- Right-click a symbol instance in the Dataflow pane and choose **Edit Symbol**, as shown in Figure 54.
- Right-click a symbol name in the Symbols pane and choose **Edit**, as shown in Figure 55.
- Create a new dataflow symbol using the Symbols pane, as described in Creating a new dataflow symbol using the Symbols pane.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

Figure 54. Entering symbol editing mode using the Dataflow pane

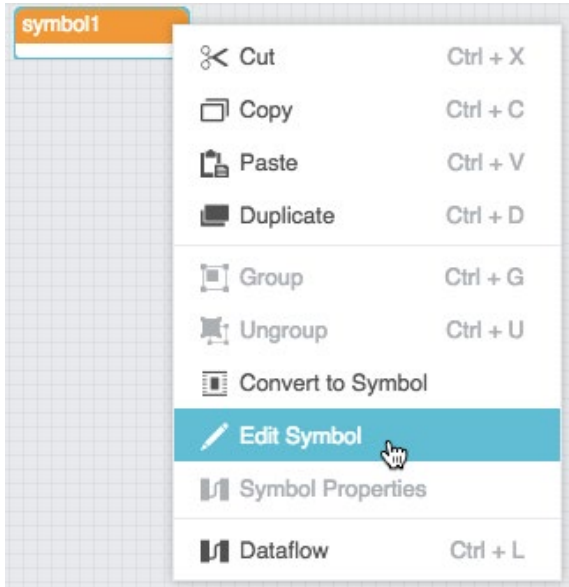
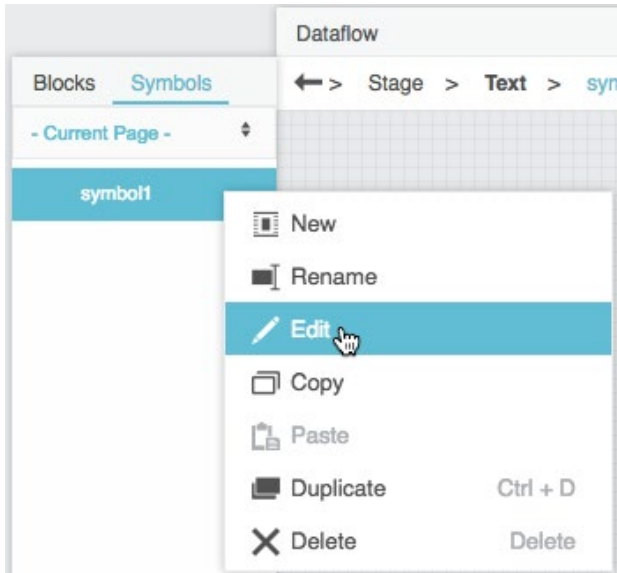


Figure 55. Entering symbol editing mode using the Symbols pane



Editing a symbol dataflow model

Changes made to a symbol dataflow model affect all instances of that symbol.

To edit a symbol dataflow model:

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

1. Make sure you have entered symbol editing mode.
2. Edit the dataflow model.

For example, you can do the following:

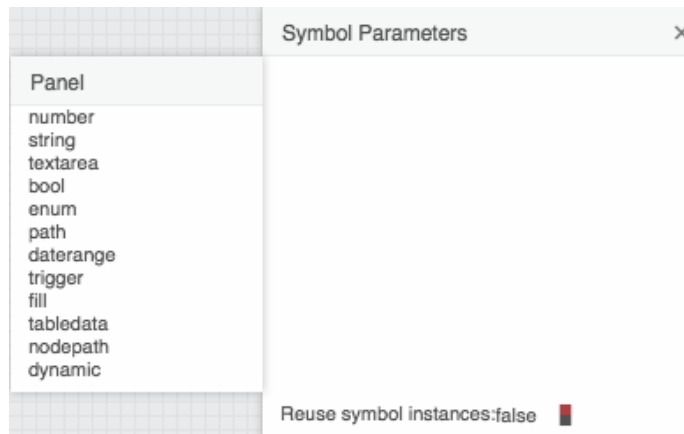
- Add blocks.
 - Rename, move, and delete blocks.
 - Edit block properties.
 - Create and delete bindings.
3. If you want to temporarily apply your changes to symbol instances, click **Apply**.
 4. Always click **OK** to save changes.

Adding symbol parameters

Symbol parameters appear as block properties for each instance. These properties can be different for each symbol instance.

To create parameters, you use a Symbol Parameters popup window. The leftmost portion of the Symbol Parameters popup window contains data types that you can add to a symbol. The rightmost portion of the Symbol Parameters popup window contains the parameters of the current symbol. Figure 56 shows the two portions of the Symbol Parameters popup window.

Figure 56. Symbol Parameters popup window



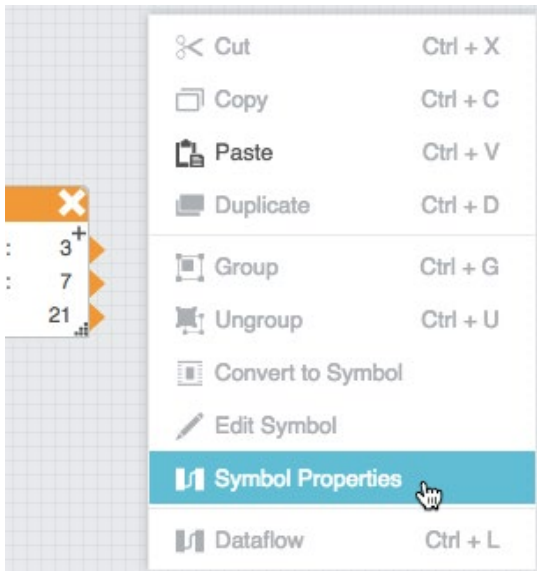
To define parameters:

1. Make sure you have entered symbol editing mode.
2. Right-click an empty area in the Dataflow pane and choose **Symbol Properties**, as shown in Figure 57.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

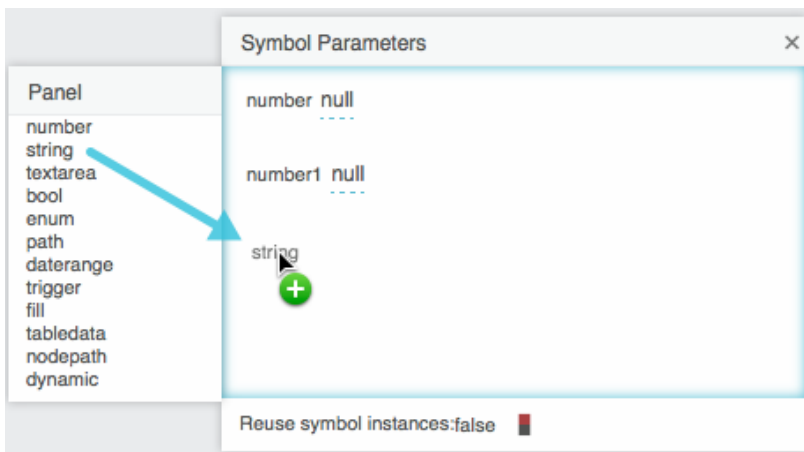
Figure 57. Accessing symbol parameters



The Symbol Parameters popup window appears.

3. Click and drag a data type and drop the data type on the rightmost portion of the popup window, as shown in Figure 58.

Figure 58. Adding symbol parameters

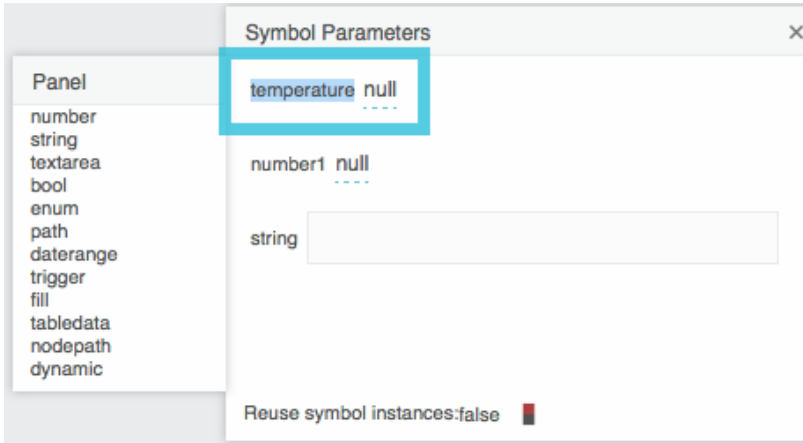


4. To change the label for a symbol parameter, double-click the label and replace it, as shown in Figure 59.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

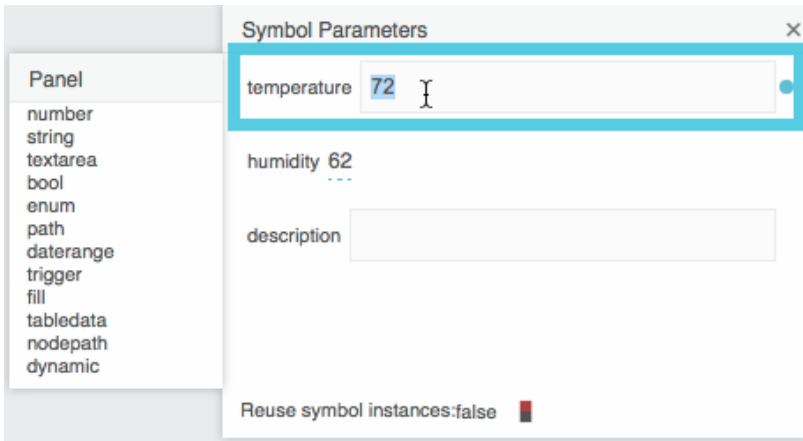
Dataflow symbols and dataflow repeaters

Figure 59. Changing parameter labels



5. To define a default value for a symbol parameter, enter the value, as shown in Figure 60.

Figure 60. Assigning a default parameter value



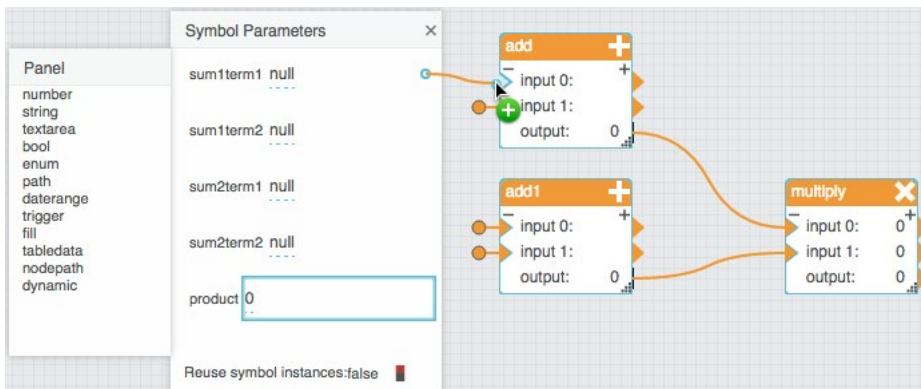
6. To cause a symbol parameter value to determine a block property inside the symbol, create a binding from the symbol parameter to the block property, as shown in Figure 61.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

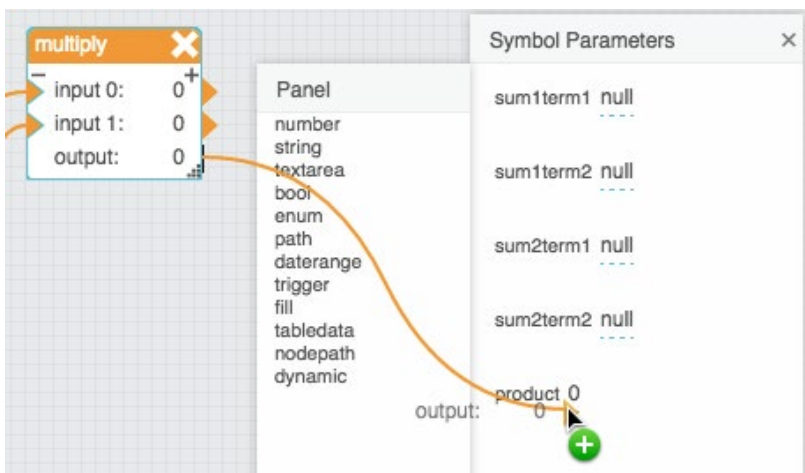
Dataflow symbols and dataflow repeaters

Figure 61. Creating a binding from a parameter to a property inside a symbol



- To cause a block property to determine a symbol parameter value, create a binding from the block property to the symbol parameter, as shown in Figure 62.

Figure 62. Creating a binding from a property inside a symbol to a parameter



- Always click **OK** to save changes.

Adding a dataflow symbol instance

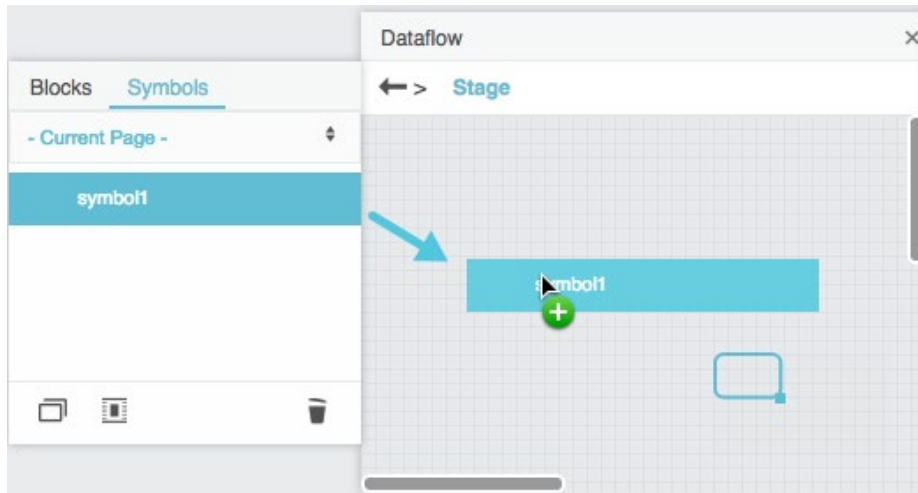
To add a dataflow symbol instance to a dataflow model:

- Open the dataflow model.
- Make sure the Symbols pane, not the Blocks pane, is selected.
- Find the symbol name in the Symbols pane.
- Drag the symbol name and drop it on the Dataflow pane, as shown in Figure 63.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

Figure 63. Adding a symbol to a dataflow model



A block appears that represents the symbol instance.

Editing symbol instance properties

By default all symbol instance properties are unpinned. You can pin the properties. See [Pinning and unpinning properties](#).

To edit the properties of a symbol instance:

1. Select the block that represents the symbol instance.
2. Edit the block properties normally.

Creating a dataflow repeater

To create a dataflow repeater:

1. Add a Repeater block to the dataflow model.
2. Bind the input table to the **data** property of the Repeater block.
3. Find the symbol in the Symbols pane.
4. Drag the symbol to the **symbol** property of the Repeater block.

Note: You can also type the symbol name in the **symbol** property field.

5. With the Repeater block selected, in the Block Properties pane, expand the **Renderer** property category. Symbol parameters appear as **Renderer** properties in the Block Properties pane.
6. With the Repeater block selected, invoke the value of the **data** property to open the source table.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

7. To bind a source table column to a Repeater property, drag the *column header* from the source table to the property.

The output table appears as the **output** property of the Repeater block.

8. To bind additional table columns, repeat step 7.

Dataflow Symbol and Repeater tutorials

These tutorials show you how to create a simple dataflow symbol and dataflow repeater.

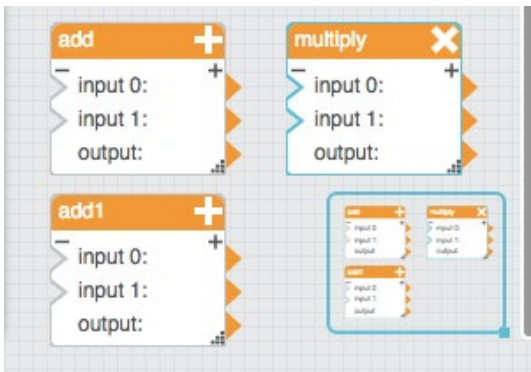
Tutorial: Multiplication dataflow symbol

This example shows you how to create a simple dataflow symbol that multiplies two sums.

To create the multiplication dataflow symbol:

1. Open any dataflow model.
2. Add a Multiply block and two Add blocks to the dataflow model, as shown in Figure 64.

Figure 64. Blocks for multiplySums symbol

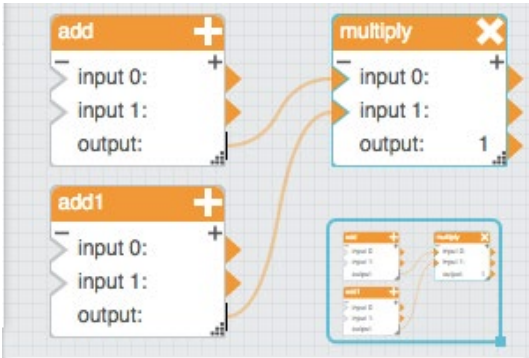


3. Bind the **output** properties of the Add blocks to the **input** properties of the Multiply block, as shown in Figure 65.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

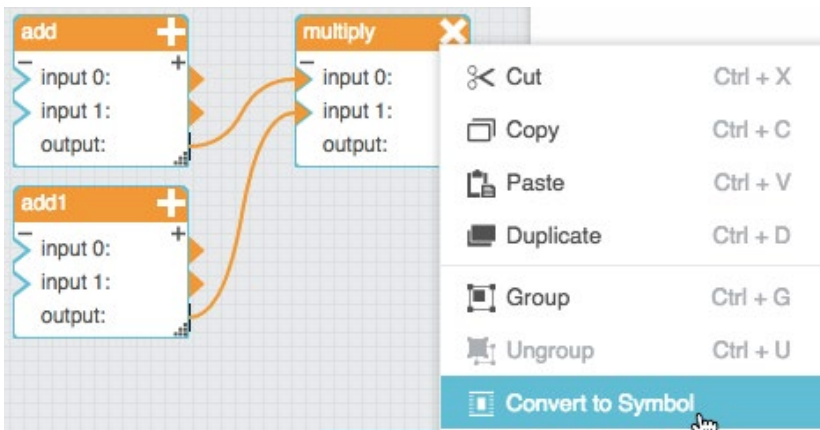
Dataflow symbols and dataflow repeaters

Figure 65. Bindings for multiplySums symbol



4. Select all three dataflow blocks.
5. Right-click one of the selected blocks and choose **Convert to Symbol**, as shown in Figure 66.

Figure 66. Converting blocks to a symbol



6. Enter `multiplySums` as the symbol name, as shown in Figure 67, and click **OK**.

Figure 67. Creating a symbol name



The three blocks are replaced by a symbol instance. In the Symbols pane, **multiplySums** appears in the symbols list.

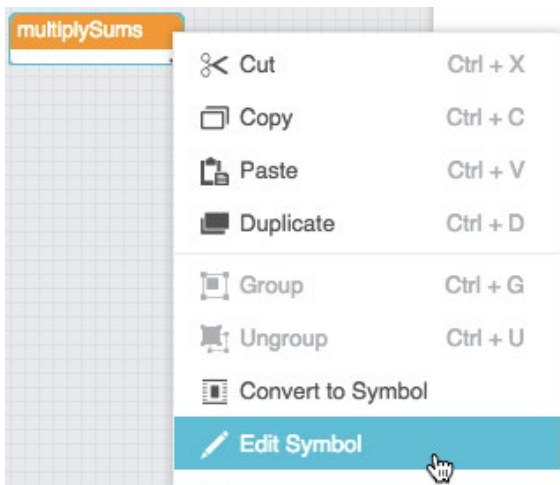
7. Right-click the symbol instance and select **Edit Symbol**, as shown in Figure 68.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

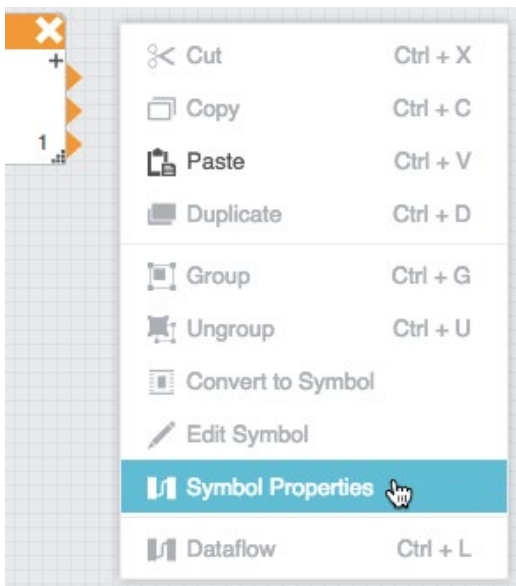
Figure 68. Accessing symbol editing mode



You enter symbol editing mode for the **multiplySums** symbol.

8. Right-click the background of the symbol editing window and select **Symbol Properties**, as shown in Figure 69.

Figure 69. Accessing symbol parameters

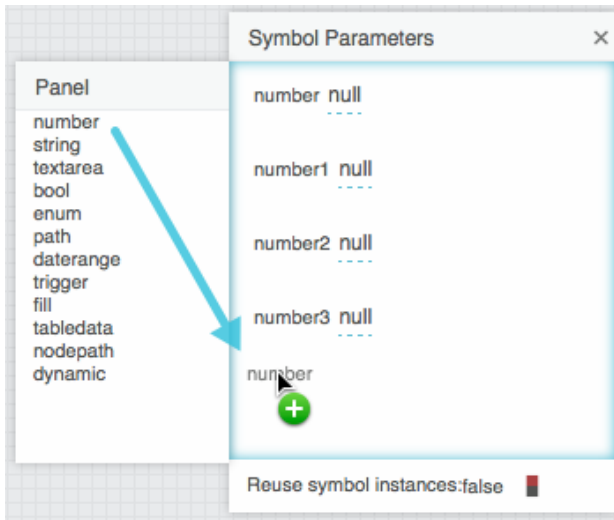


9. Drag five **number** parameters to the rightmost portion of the Symbol Parameters popup window, as shown in Figure 70.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

Figure 70. Symbol parameters for multiplySums symbol

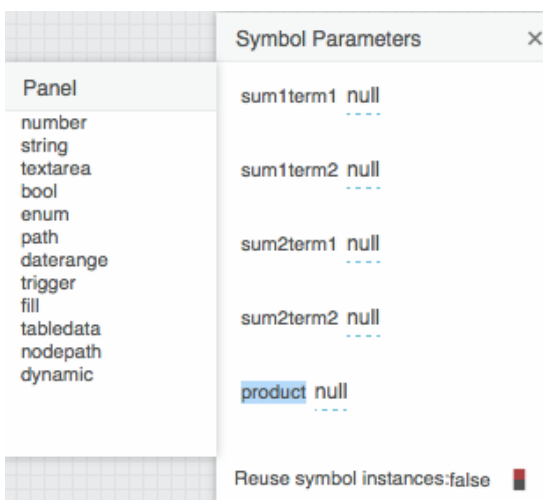


10. Edit the labels of the five parameters to the following labels:

- sum1term1
- sum1term2
- sum2term1
- sum2term2
- product

See Figure 71.

Figure 71. Parameter labels for multiplySums symbol



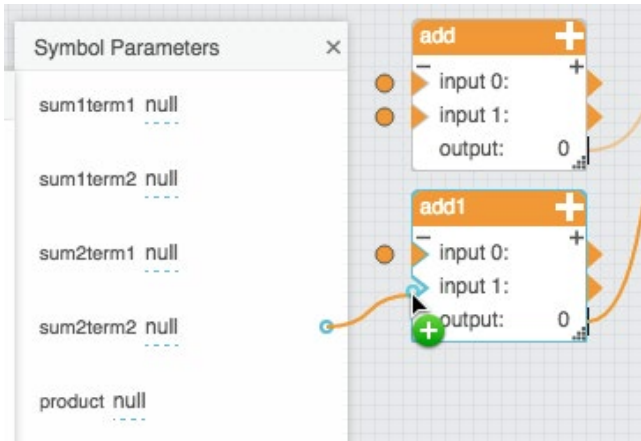
Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

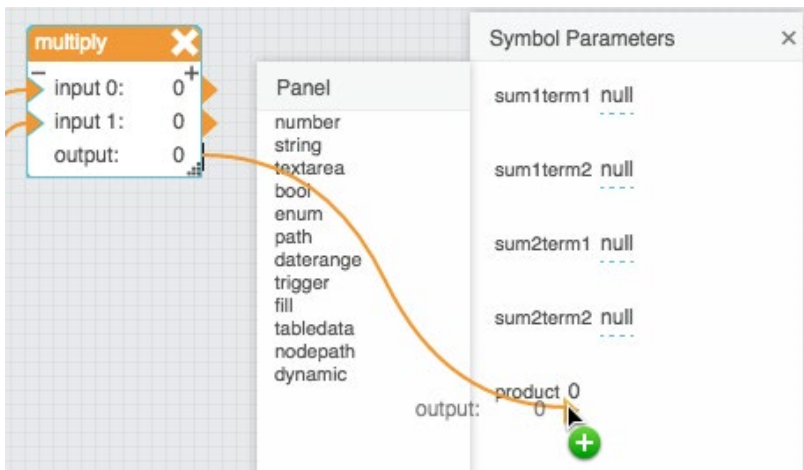
- Bind the first four symbol parameters to the **input** properties of the **Add** blocks, as shown in Figure 72.

Figure 72. Sum terms bindings for multiplySums symbol



- Bind the **output** property of the **Multiply** block to the **product** symbol parameter, as shown in Figure 73.

Figure 73. Product binding for multiplySums symbol



- Click **OK** to exit symbol editing mode.
- In the Dataflow pane, select the **multiplySums** symbol instance.

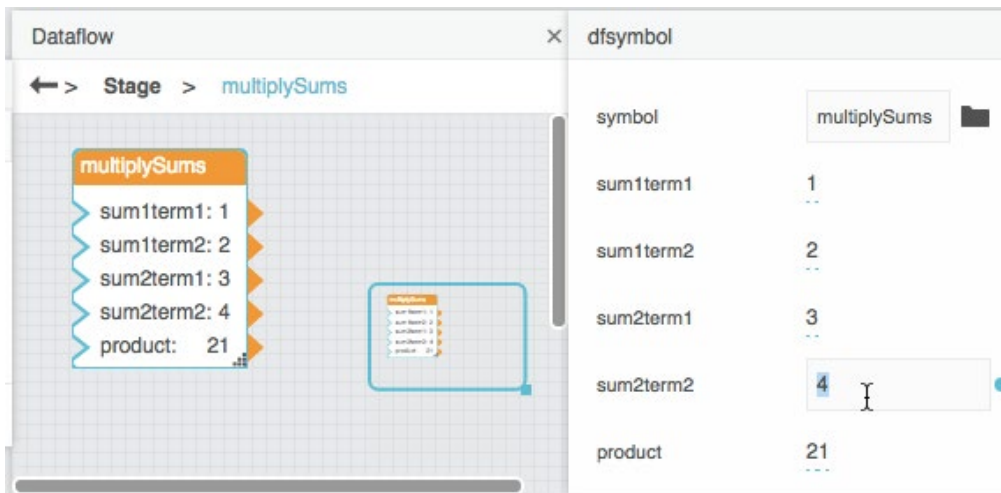
The parameters of the **multiplySums** symbol appear in the Block Properties pane.
- To cause properties to appear in the visual block in the Dataflow pane, pin the properties. See Pinning and unpinning properties.
- To test your symbol, enter numbers for the first four parameters.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

The **product** property updates. See Figure 74.

Figure 74. Instance of multiplySums symbol



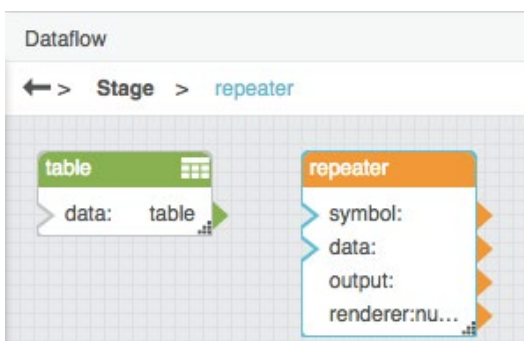
Tutorial: Multiplication dataflow repeater

This example uses the dataflow symbol created in the previous tutorial to create a simple dataflow repeater.

To create the multiplication dataflow repeater:

1. Create the **multiplySums** dataflow symbol as described above.
2. Open any dataflow model.
3. Add a Table block and a Repeater block to the dataflow model, as shown in Figure 75.

Figure 75. Blocks for multiplySums repeater

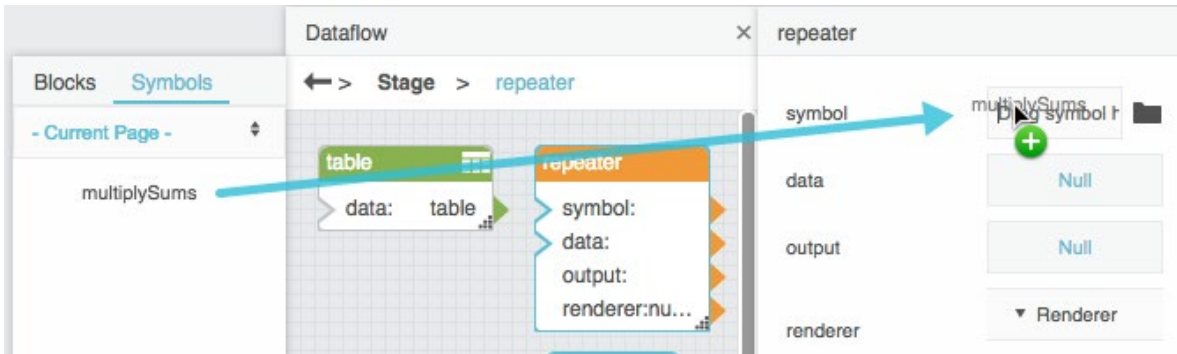


4. With the Repeater block selected, open the Symbols pane and drag the **multiplySums** symbol name to the **symbol** property, as shown in Figure 76.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

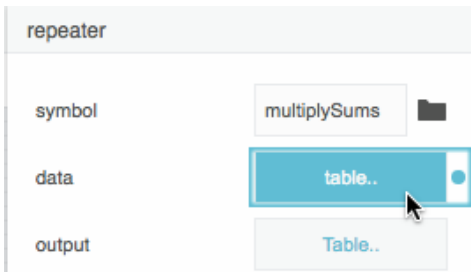
Dataflow symbols and dataflow repeaters

Figure 76. Defining the symbol for a dataflow repeater



5. Select the Table block and invoke the value of the **data** property to open the table, as shown in Figure 77.

Figure 77. Viewing table data

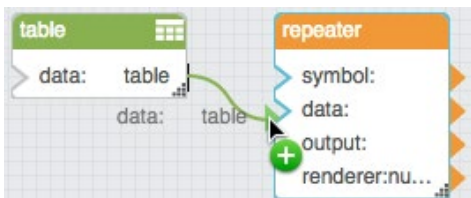


6. Use the String and CSV Parser blocks to edit the table so that the table contains four columns of numbers. See Figure 78.

Figure 78. Table for multiplySums repeater

row	v1	v2	v3	v4
0	1	4	4	5
1	3	2	5	4
2	2	6	6	3

7. Bind the **data** property of the Table block to the **data** property of the Repeater block.



Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow symbols and dataflow repeaters

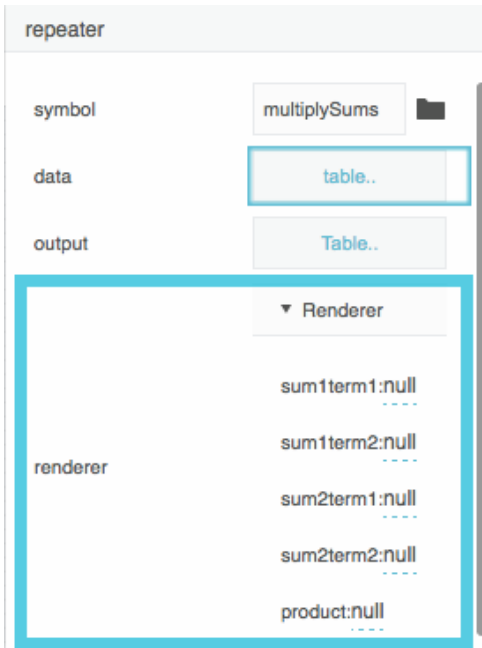
8. With the Repeater block selected, invoke the value of the **data** property to open the table. See Figure 79.

Figure 79. Viewing the input table for a repeater



9. Expand the **Renderer** section in the Block Properties pane, as shown in Figure 80.

Figure 80. Expanded renderer properties for multiplySums repeater

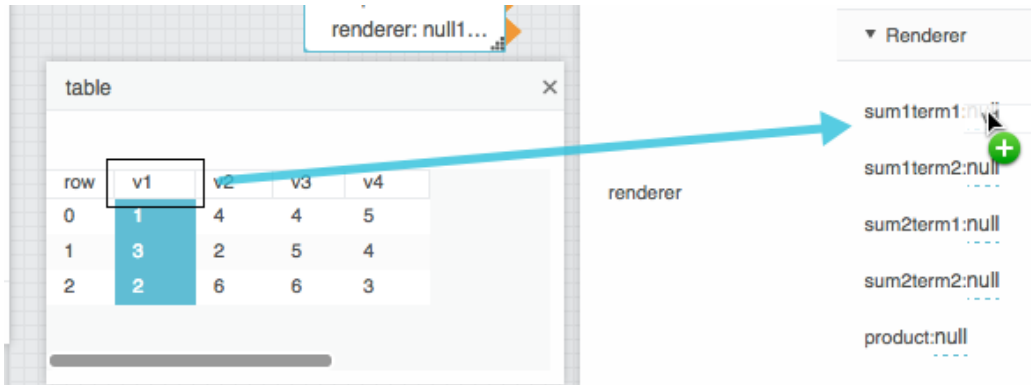


10. To bind the table columns to the first four **Renderer** properties, drag and drop each column header, as shown in Figure 81.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

Figure 81. Binding column headers to renderer properties



11. To view the output of the repeater, invoke the value of the **output** property, as shown in Figure 82.

Figure 82. Viewing the output table for a repeater



The output table contains a row for each input table row. It contains a column for each symbol property, including the **product** property. Figure 83 shows the output table.

Figure 83. Output table for multiplySums repeater

row	sum1term1	sum1term2	sum2term1	sum2term2	product
0	1	4	4	5	45
1	3	2	5	4	45
2	2	6	6	3	72

Dataflow blocks

This section provides details about the purpose and properties of each dataflow block.

Variables Blocks

String

The String block holds a value as a sequence of characters.

The String block's value can include line breaks. Press Enter to create a line break.

Input/Output Property

The following property of the String block can take input and give output.

value (string) Sets and returns the value held by this String block.

Examples

Figure 84 demonstrates two String blocks.

The leftmost block holds the following string: Hello World!

The rightmost block holds the following string: Asset 1

Figure 84. String block



Number

The Number block holds a value that describes a measurable quantity as a number.

The value can be positive or negative.

Input/Output Property

The following property of the Number block can take input and give output.

value (number) Sets and returns the value held by this Number block.

Examples

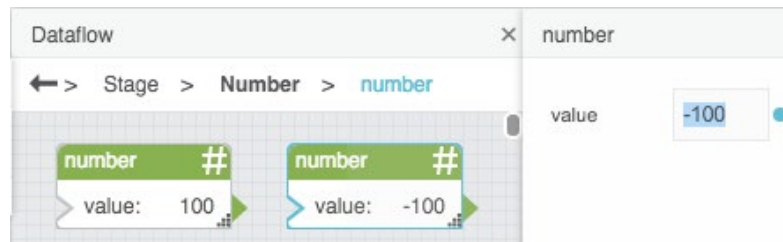
Figure 85 demonstrates two Number blocks. The leftmost Number block holds the number one hundred (100). The rightmost Number block holds the number negative one hundred (-100).

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 85. Number Block



Boolean

The Boolean block holds a value that can be only TRUE or FALSE.

Input/Output Property

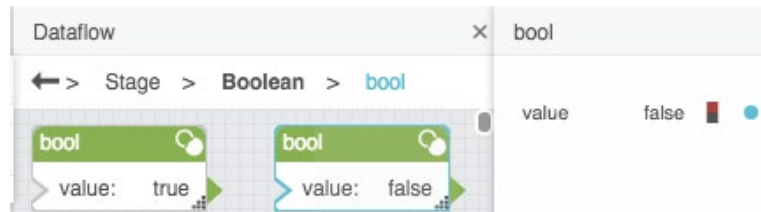
The following property of the Boolean block can take input and give output.

value (Boolean) Sets and returns the value held by this Boolean block.

Examples

Figure 86 demonstrates two Boolean blocks. The leftmost Boolean block holds a TRUE value. The rightmost Boolean block holds a FALSE value.

Figure 86. Boolean Block



Table

The Table block holds values as a set of records.

Input/Output Properties

The following properties of the Table block can take input and give output.

data (table) Sets and returns the table held by this Table block.

save (trigger) Removes any binding to the **data** property. When a binding is deleted by the **save** trigger, data currently in the table does not change, even if that data was determined by a binding. See Unbinding and saving table data.

Data Services Blocks

List Node

The List Node block returns metadata about a node and its children. The List Node block stays up to date even if the children of the node change during the session.

Input/Output Properties

The following properties of the List Node block can take input and give output.

path (string)	Specifies the absolute location of the node to get metadata for. Drag the relevant node from the Data pane to this field.
mode (enum)	Determines which children of this node are listed. The value of the mode property can be one of the following: <ul style="list-style-type: none"> all—All children are listed. points—Only nodes that represent data points are listed. actions—Only nodes that represent actions are listed. nodes—Only nodes that are not points or actions are listed.
resolveActions (Boolean)	Determines whether to create an output table that describes available actions for this node's children. By default, this property is disabled.

Output Properties

The following properties of the List Node block can give output but cannot take input.

name (string)	Returns the name of the node.
children (table)	Returns a table that lists the children of the node.
metadata (table)	Returns a table that lists metadata about the node.
actions (table)	Returns a table that describes the actions that can be taken on the node, such as setting the node's value. Each row of this table contains a nested table that lists the parameters of the action. To view a nested table, see Viewing the contents of nested tables.
error (string)	Returns the error message, if any.

Example

Figure 87 shows, clockwise from top left:

- A List Node block

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

- The **metadata** table for this node
- The **children** table for this node

Figure 87. List Node Block

The screenshot shows the configuration for the **listNode** block. The configuration is as follows:

```

service:
path: /downstream/System
mode: all
resolveActions:
name: System
profile: dsa/link
children: Table
metadata: Table
actions:
error:
  
```

Below the configuration, there are two tables. The first is a small table with 2 rows and 3 columns:

row	name	value
0	\$\$dsid	System-UP-R-e80JYa
1	\$is	dsa/link

The second table is a larger table with 8 columns and 5 rows:

row	name	path	base	profile	type	invokable	writable
1	CPU Usage	/downstr...	/downstr...	node	number		
2	Diagnostics Mode	/downstr...	/downstr...	diagnosticsMode	bool[disabled...		write
3	Disk Usage	/downstr...	/downstr...	node	number		
4	Execute Command	/downstr...		executeComma...			write

Get Default Parameters

The Get Default Parameters block returns the default parameters and parameter values associated with a data action. See Invoke Action.

Input/Output Properties

The following properties of the Get Default Parameters block can take input and give output.

- path** (string) Specifies the absolute location of the node where the data action is located.
- action** (string) Specifies the name of the data action to get default parameters for.

Additional Properties

When the **action** property is populated, the Get Default Parameters block acquires additional properties. These additional properties correspond to the parameters of the action specified. The additional properties hold the default values of the parameters.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Using an Invoke Action block to easily get the path and action

To easily get the path and action:

1. In the Data or Metrics pane, right-click the node to see the list of available actions.
2. Choose the relevant action and drag it to the Dataflow pane.

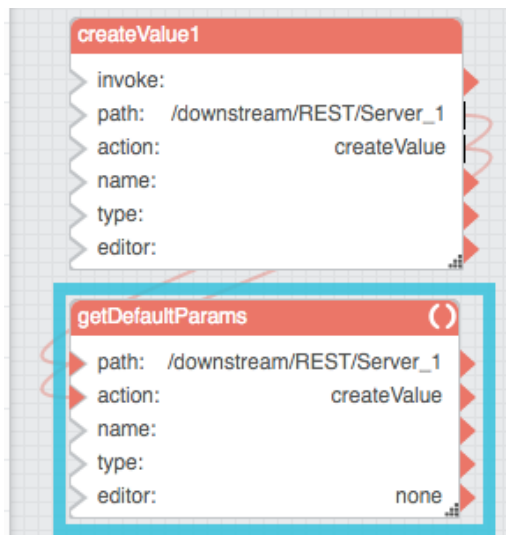
This creates an Invoke Action block.

3. Bind the **path** and **action** properties from the Invoke Action block to the Get Default Parameters block.

Example

Figure 88 demonstrates a typical use of the Get Default Parameters block. In this example, the Get Default Parameters block reports three parameters for the chosen action: **name**, **type**, and **editor**. The **name** and **type** parameters are null by default and the **editor** parameter has a default value of “none.”

Figure 88. Get Default Parameters Block



String Loader

The String Loader block accesses the specified URL and either returns the string located in that file or performs other actions based on HTTP request methods. The target domain must allow this access in order for the String Loader block to work.

The String Loader block is commonly used to load the contents of JSON or CSV files. You can link its output to a JSON Parser or CSV Parser block to create a table from the source file.

Input/Output Properties

The following properties of the String Loader block can take input and give output.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

invoke (trigger)	Causes the string to be loaded. The invoke property works only if enabled is set to TRUE.
enabled (Boolean)	Specifies whether the string loader is enabled. TRUE —The string loads as specified by invoke , autoRun , and interval . FALSE —The string does not load.
interval (number)	Specifies how often the string is automatically reloaded, in seconds. A value of 0 means that the string is reloaded on autoRun or invoke only.
autoRun (Boolean)	Specifies whether the string is loaded automatically. TRUE —The string is loaded every time any property is changed. The first time that the path property is populated counts as a change. FALSE —The script runs only when the invoke property is triggered or at the specified interval.
timeout (number)	Specifies how long the block attempts to load the string before the request is canceled.
method (enum)	Specifies the HTTP request method . In almost all cases, you want GET or POST. GET —Retrieves the string at the specified URL. POST —Adds something to the string at the specified URL.
headers (string)	Sets the headers for the HTTP request. Use the following syntax: <code><header1>=<value1>&<header2>=<value2></code>
sendCookies (Boolean)	Specifies whether to send a cookie to the URL when the string gets retrieved. There is a security risk involved when sending cookies. However, you must send cookies in order to get certain custom information.
path (URL)	Specifies the URL to load the string from.
data (string)	Sets and returns either the payload for a POST request or the string to be added to the URL for a GET request.
errorGroup (string)	See Catch Error.

Output Properties

The following properties of the String Loader block can give output but cannot take input.

loading (Boolean)	Returns whether the string is currently being loaded.
--------------------------	---

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

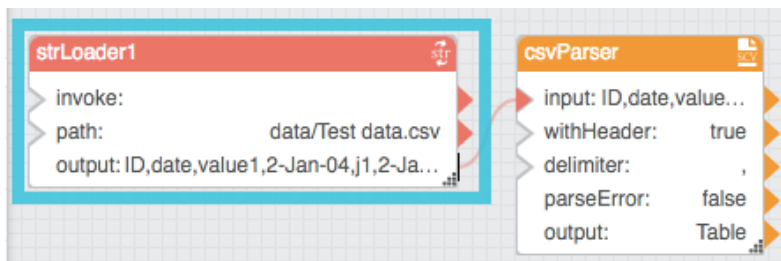
Dataflow blocks

respHeaders (string)	Returns the HTTP response headers that are received when the string is loaded.
status (string)	Returns the HTTP response status code .
output (string)	Returns the string contained in the file.
onComplete (event)	Fires when the string finishes loading.
error (string)	Returns the error message, if any.

Example

Figure 89 demonstrates a typical use of the String Loader block. In this example, the String Loader block retrieves the contents of a CSV file. Then, the String Loader block passes the file contents to a CSV Parser block.

Figure 89. String Loader Block



String Uploader

The String Uploader block saves a string as a new file. If a file with the specified name and extension already exists, this block replaces the contents of the existing file.

Input/Output Properties

The following properties of the String Uploader block can take input and give output.

invoke (trigger)	Causes the string to be uploaded. The invoke property works only if enabled is set to TRUE.
enabled (Boolean)	Specifies whether the String Uploader block is enabled. TRUE —The string is uploaded when this block is invoked. FALSE —Nothing happens when this block is invoked.
path (string)	Specifies where to upload the string. Must include a file name and extension.
data (string)	Holds the string that gets uploaded.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

- deleteEmptyFile** (Boolean) Specifies what happens when the **data** property is null.
- TRUE**—If **data** is null, no new file is created. If **data** is null and the file already exists, the file is deleted.
- FALSE**—If **data** is null, a file is created when this block is invoked. The file is empty. If **data** is null and the file already exists, the file's contents are deleted but the file remains.

Output Properties

The following properties of the String Uploader block can give output but cannot take input.

- onComplete** (event) Fires when the string has been uploaded.
- error** (string) Returns the error message, if any.

Get Children

The Get Children block returns a table that lists the children of a data node.

Note: You can drag a node from the Data pane into the Dataflow pane to automatically create a Get Children block for that node.

Input/Output Property

The following property of the Get Children block can take input and give output.

- path** (string) Specifies the node for which to list children. You can drag the node from the Data pane to this field.

Output Properties

The following properties of the Get Children block can give output but cannot take input.

- output** (table) Returns a table that lists the children of the specified node. The table includes the following fields:
- name**—The name of the child node.
- path**—The full path of the child node.
- hasChildren**—TRUE if the child node has children, FALSE if the child node is terminal.
- hasValue**—TRUE if the child node is associated with a value, FALSE otherwise.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

hasHistory—TRUE if the child node is associated with a value history, FALSE otherwise.

type—The data type of the child node, if applicable.

icon—The absolute file path to the icon that represents the child node, if applicable.

unit—The units of the child node, if applicable.

enum—The names of the enum options, if applicable. For a Boolean value, this column holds the strings associated with each value, such as On and Off.

error (string) Returns the error message, if any.

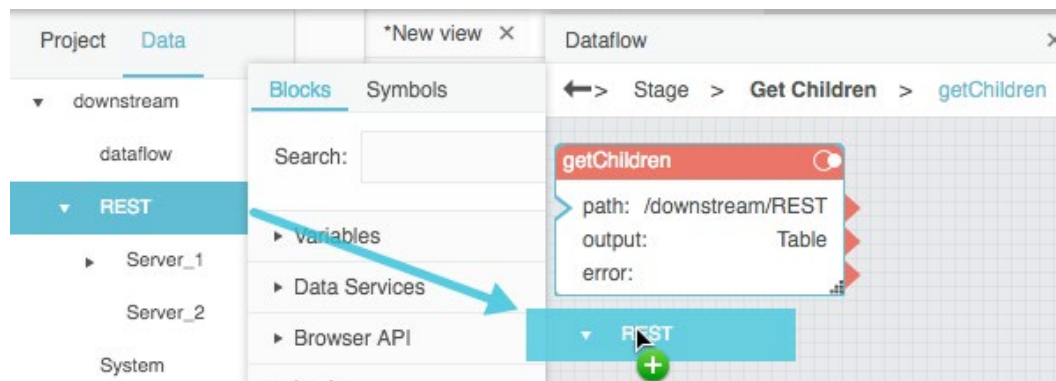
About the Output Table

The output table of the Get Children block is only one level deep. Therefore, when a node that has children is listed, the node is represented with a name, not a table. You can use multiple Get Children blocks to retrieve deeper lists.

Example

Figure 90 demonstrates how to create a Get Children block by dragging a node to the Dataflow pane.

Figure 90. Get Children Block



Get Node

The Get Node block returns metadata about a data node.

Input/Output Properties

The following properties of the Get Node block can take input and give output.

path (string) Specifies the absolute location of the node to get metadata for.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

Output Properties

The following properties of the Get Node block can give output but cannot take input.

name (string)	Returns the name of the node.
hasChildren (Boolean)	Returns TRUE if the node has children, FALSE if it is terminal.
hasValue (Boolean)	Returns TRUE if the node is associated with a value, FALSE otherwise.
hasHistory (Boolean)	Returns TRUE if the node is associated with a value history, FALSE otherwise.
type	Returns the data type of the node, if applicable.
icon (string)	Returns the absolute file path to the icon that represents the node.
unit (string)	Returns the units of the node, if applicable.
enum (string)	Returns the names of the enum options, if applicable. For a Boolean value, this column holds the strings associated with each value, such as On and Off.
actions (table)	Returns a table of the actions that can be taken on this node, such as setting its value. Each row of this table contains a nested table that lists the parameters of that action. To view a nested table, see How do I view the data in nested tables?
error (string)	Returns the error string, if applicable.

Example

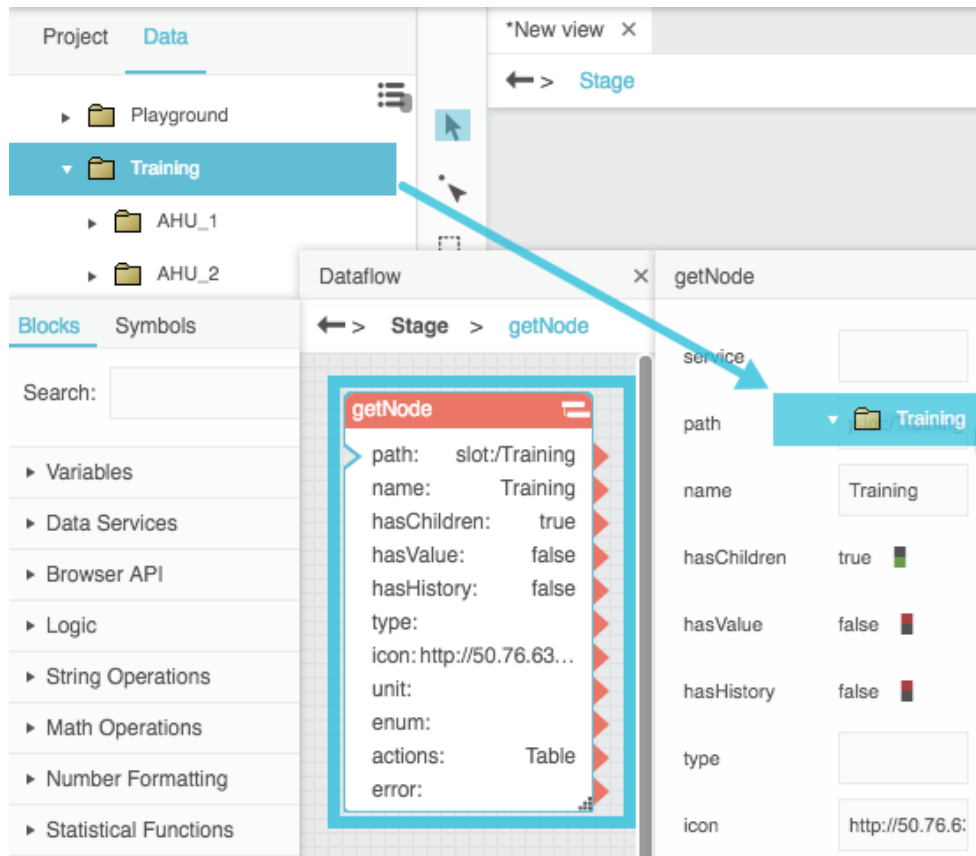
Figure 91 shows an example of the Get Node block. In this example, the **Training** node is dragged to the **path** property to populate the Get Node block.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 91. Get Node Block



Multi-Histories

The Multi-Histories block retrieves a table of information about the value histories of multiple data nodes.

Note: The **invoke** property must be triggered to load the table the first time and also to load the table after any changes, unless **interval** is specified or **autoRun** is enabled.

Input/Output Properties

The following properties of the Multi-Histories block can take input and give output.

- invoke** (trigger) Causes the histories to be loaded. This trigger works only if **enabled** is set to TRUE.
- enabled** (Boolean) Specifies whether this Multi-Histories block is enabled.
 - TRUE**—The histories load as specified by **invoke**, **autoRun**, and **interval**.
 - FALSE**—The histories do not load.

Dataflow blocks

interval (number)	Specifies how often the histories are loaded, in seconds. A value of 0 means that the histories are reloaded on autoLoad or invoke only.
autoRun (Boolean)	Specifies whether the histories are loaded automatically. TRUE —The histories are loaded every time any property of this block is changed. The first time that the path property is populated counts as a change. FALSE —The histories are loaded when the invoke property is triggered, or at the specified interval .
timeout (number)	Specifies, in seconds, how long this block attempts to load the histories before the request is canceled.
paths (multi-line text)	Specifies the list of paths to the histories to load, separated by line breaks. Do not use commas to separate entries. You can drag metrics from the Metrics pane to this field. Press Alt + Enter to create line breaks.
timeRange (date range)	Specifies the date and time range to return the histories for. You can click the button and use the date range picker to specify a range.
dataInterval (enum)	Specifies the date or time interval between each consecutive pair of data points.
showStatus (Boolean)	Determines whether a status column is included for each value history column in the output table. Status values report the status of the node at the time that data was collected, such as “down” or “overridden.”
rollup (enum)	Specifies the type of rollup used. The value of the rollup property can be one of the following: Avg —The average of all number values in the interval. Min —The smallest number value in the interval. Max —The greatest number value in the interval. Sum —The sum of all number values in the interval. First —The first value in the interval. Last —The last value in the interval. Count —How many non-null values are in the interval.

Dataflow blocks

missingValue (enum)	<p>Specifies what value to display in the event of missing data. The value of the missingValue property can be one of the following:</p> <p>Null—A null value.</p> <p>Use previous—The value for the previous timestamp.</p> <p>Use next—The value for the next timestamp.</p> <p>Interpolate—The average between the previous and next timestamps.</p> <p>Zero—Zero.</p>
header (enum)	<p>Specifies the string that is used as the name of each value history column in the output table. The value of the header property can be one of the following:</p> <p>Default—The default string: vn</p> <p>Name—The name of this node, for which this column is a value history.</p> <p>Parent—The name of this node's parent.</p> <p>Parent_Name—The names of both this node and its parent.</p>
delta (enum)	<p>Specifies what is in the value history columns of the output table. The value of the delta property can be one of the following:</p> <p>None—The value collected at the given time.</p> <p>With previous—The difference between the values for this interval and for the previous one.</p> <p>With next—The difference between the values for this interval and for the next one.</p>

Output Properties

The following properties of the Multi-Histories block can give output but cannot take input.

loading (Boolean)	Returns whether the histories are currently loading.
output (table)	<p>Returns a table of the given histories' values. The columns of this table are as follows:</p> <p>timestamp—The time associated with a value.</p> <p>vn—The first data node's value or its delta if specified by the delta property.</p> <p>status—The status of the data node at the time of collection.</p>

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Adding paths to the Multi-Histories block

To add paths:

1. Within the **paths** field, click the **Edit in Window** icon to open a **loadHistories.paths** popup window.
2. In the Data pane, choose the relevant device or data source.
3. Drag the relevant **History** icon from the Metrics pane to the **loadHistories.paths** popup window.
4. Press Enter to create a new line.
5. Repeat Steps 2 to 4 until all paths are entered.

Example

Figure 92 demonstrates how to populate the **paths** property by dragging histories from the Metrics pane.

Figure 92. Multi-Histories Block



Zip Parser

The Zip Parser block takes a ZIP file as input and returns a table that contains the ZIP file's contents.

Input/Output Property

The following property of the Zip Parser block can take input and give output.

data (binary object ZIP file) Receives a ZIP file, specified as a binary object.

Output Property

The following property of the Zip Parser block can give output but cannot take input.

output (table) Returns a table that contains the files that comprise the input ZIP file.

Example

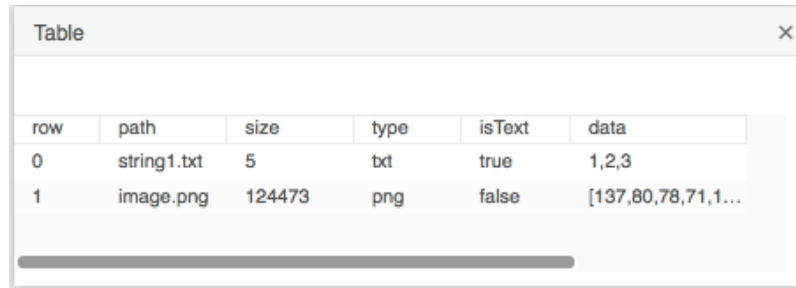
Figure 93 demonstrates an example of a table returned by the Zip Parser block.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 93. Table returned by a Zip Parser block



row	path	size	type	isText	data
0	string1.txt	5	txt	true	1,2,3
1	image.png	124473	png	false	[137,80,78,71,1...

Load Value

The Load Value block retrieves information about the value of a data node.

To automatically create a Load Value block for a node at a particular path, you can drag that node from the Metrics pane into the Dataflow pane.

Input/Output Properties

The following properties of the Load Value block can take input and give output.

path (string)	Specifies the node for which to return information.
enabled (Boolean)	Specifies whether the Load Value block updates when the node's value changes.
qos (enum)	<p>Determines persistence. For a qos of 1, 2, or 3, you can access table cells by binding the value property to a Table block.</p> <p>0—The real-time value is loaded and is replaced with new values in real time.</p> <p>1—The real-time value is loaded, is output in a table and is replaced with new values in real time. If more than one value is loaded in a request, multiple rows appear in the table. The output is intended to be used with a threaded dataflow repeater in "stream" mode.</p> <p>2—Intended only for remote dataflow. The real-time value is loaded, is output in a table, and is replaced with new values in real time. Only one row appears in the table. If the connection is lost, values persist on the server and the most recent value is sent to the client when the connection is restored.</p> <p>3—Intended only for remote dataflow. The real-time value is loaded, is output in a table, and is replaced with new values in real time. If more than one value is loaded in a request, multiple rows appear in the table. If the connection is lost, values persist on the server and all values stored on the server are sent to the client when the connection is restored. The output is intended to be used with a threaded dataflow repeater in "stream" mode.</p>

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Output Properties

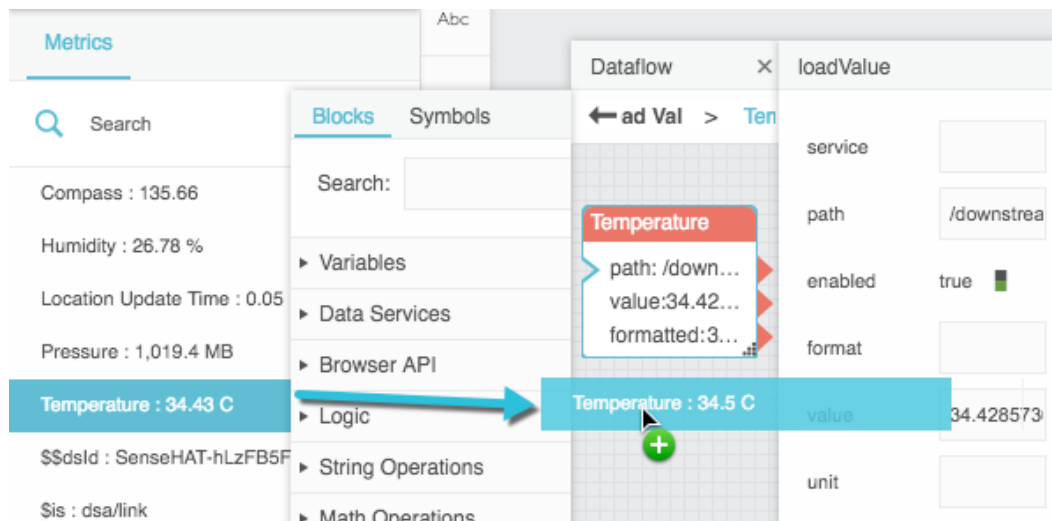
The following properties of the Load Value block can give output but cannot take input.

value	Returns the current value of the node, if enabled is TRUE. Returns the last collected value of the node, if enabled is false.
unit (string)	Returns the units of the node.
formatted (string)	Returns the formatted value. For example, for a numeric value, formatted returns a string that includes units. For an enum or Boolean value, it returns the string associated with the value.
status (string)	Returns the status of the node, if applicable, such as “down” or “overridden.”
error (string)	Returns the error message, if any.
lastUpdate (string)	Returns a string that indicates when the node was last updated.

Example

Figure 94 demonstrates how to create a Load Value block by dragging a node to the Dataflow pane.

Figure 94. Load Value Block



Load History

The Load History block retrieves information about the value history of a data node.

To automatically create a Load History block for a node at a particular path, you can drag that node's History icon from the Metrics panel into the dataflow window.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

Note: The **invoke** property must be triggered to load the table the first time and also to load the table after any changes, unless **interval** is specified or **autoRun** is enabled.

Input/Output Properties

The following properties of the Load History block can take input and give output.

invoke (trigger)	Causes the history to be loaded. The invoke property only works if enabled is set to TRUE.
enabled (Boolean)	Specifies whether the Load History block is enabled. TRUE —The history loads as specified by invoke , autoRun , and interval . FALSE —The history does not load.
interval (number)	Specifies how often the history is loaded, in seconds. A value of 0 means that the history is reloaded on autoLoad or invoke only.
autoRun (Boolean)	Specifies whether the history is loaded automatically. TRUE —The history is loaded every time any property of this block is changed. The first time that the path property is populated counts as a change. FALSE —The history is loaded only when the invoke property is triggered or at the specified interval .
timeout (number)	Specifies how long the loader attempts to load the history before the request is canceled.
path (string)	Specifies the history for which to return information.
timeRange (date range)	Specifies the date and time range to return the history for. You can click the button and use the date range picker to specify a range.
dataInterval (enum)	Specifies the date or time interval between each consecutive pair of data points.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

- rollup** (enum) Specifies the type of rollup used. The value of the **rollup** property can be one of the following:
- Avg**—The average of all number values for the interval.
 - Min**—The smallest number value for the interval.
 - Max**—The greatest number value from the interval.
 - Sum**—The sum of all number values for the interval.
 - First**—The first value for the interval.
 - Last**—The last value for the interval.
 - Count**—The number of non-null values that exist for the interval.
- header** (enum) Specifies which string is used to name the **value** column in the output table. The value of the **header** property can be one of the following:
- Default**—The default string: value
 - Name**—The name of the data node.
 - Parent**—The name of the data node's parent.
 - Parent_Name**—The names of both this node and its parent.
- delta** (enum) Specifies what is in the **value** column of the output table. The value of the **delta** property can be one of the following:
- None**—The value collected at the given time.
 - With previous**—The difference between the values for this interval and for the previous one.
 - With next**—The difference between the values for this interval and for the next one.

Output Properties

The following properties of the Load History block can give output but cannot take input.

- loading** (Boolean) Returns whether the history is currently loading.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

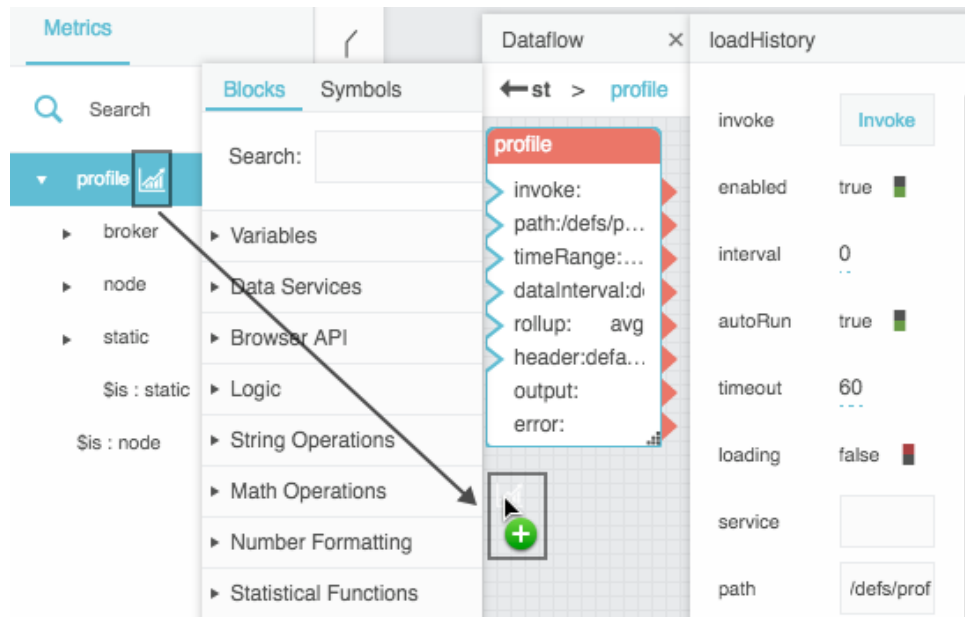
Dataflow blocks

output (table)	Returns a table of the given history's values. The columns of this table are as follows: <ul style="list-style-type: none"> timestamp—The time when the data was collected. value—The data node's value or the delta if specified. status—The status of the data node at the time of collection.
error (string)	Returns the error string, if any.

Example

Figure 95 demonstrates how to create a Load History block by dragging a **History** icon to the Dataflow pane.

Figure 95. Load History Block



Invoke Action

The Invoke Action block performs an action on a data node or data metric.

Caution: The Invoke Action block writes to the data source.

Adding an Invoke Action block to a dataflow model

This block does not appear in the Blocks pane.

To add an Invoke Action block:

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

1. In the Data or Metrics pane, right-click the node to see the list of available actions.
2. Choose the relevant action and drag it to the Dataflow pane.

This creates an Invoke Action block.

Input/Output Properties

The following properties of the Invoke Action block can take input and give output.

invoke (trigger)	Causes the action to be performed. The invoke property works only if enabled is set to TRUE.
enabled (Boolean)	Determines whether the Invoke Action block is enabled. TRUE —The action is performed as specified by invoke , autoRun , and interval . FALSE —The action is not performed.
interval (number)	Specifies how often the action is automatically performed, in seconds. A value of 0 means that the action is performed on autoRun or invoke only.
autoRun (Boolean)	Specifies whether the action is performed automatically. TRUE —The action is performed every time any property of this block is changed. The first time that the path property is populated counts as a change. FALSE —The action is performed only when the invoke property is triggered or at the specified interval.
timeout (number)	Specifies how long the block attempts to perform the action before the request is canceled.
refresh (trigger)	Reads the action's parameter list again. By default, the block reads the parameter list from the server only once, unless the path changes.
path (string)	Specifies the data node or data metric on which to perform the action. This field is automatically populated when the block is created, but it can be edited.
action (string)	Specifies the action to perform. This field is automatically populated when the block is created, but it can be edited.
getTable (Boolean)	Specifies whether to return the results of the action as a table. TRUE —The results are returned as a table. FALSE —The results are returned as properties of the Invoke Action block.
bufferSize (number)	Specifies the number of rows that can be held by the output table.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

errorGroup (string) See Catch Error.

Additional Properties

When the **action** property is populated, the Invoke Action block acquires additional properties. These additional properties are the same as the parameters of the action.

Output Properties

The following properties of the Invoke Action block can give output but cannot take input.

loading (Boolean)	Returns whether the action is currently being performed.
onComplete (event)	Fires when the action is completed.
onError (event)	Fires if an error occurs.
error (string)	Returns the error message, if any.

Logic Blocks

And

The And block is an If block. When the And block is created, the block's **op** property is set as the "and" operator. See If.

Or

The Or block is an If block. When the Or block is created, the block's **op** property is set as the "or" operator. See If.

Not

The Not block reverses the value of its input. This block returns FALSE when its input evaluates to TRUE and returns TRUE when its input evaluates to FALSE.

Input/Output Property

The following property of the Not block can take input and give output.

input (Boolean)	Receives a value that can be evaluated to TRUE or FALSE.
------------------------	--

Output Properties

The following properties of the Not block can give output but cannot take input.

output (Boolean)	Returns the opposite of the input value.
-------------------------	--

Kinetic - Edge & Fog Fabric Processing Module

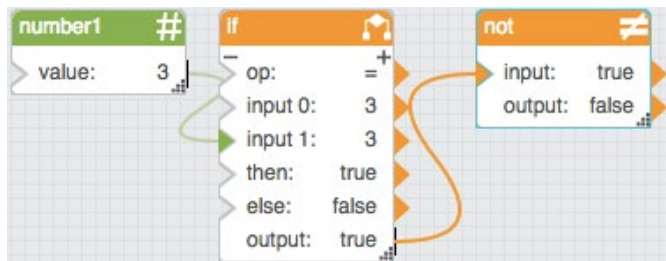
Dataflow Editor User Guide

Dataflow blocks

Examples

The following image demonstrates an example of the Not block. In this example, the Not block returns the opposite of the result of the If block.

Figure 96. Not Block



If

The If block returns one value if a condition that you specify evaluates to TRUE and another value if the condition evaluates to FALSE.

Input/Output Properties

The following properties of the If block can take input and give output.

- op** (enum) Specifies the arithmetic, comparison, or text operator to apply to the input values. It is used in conjunction with **input *n*** to form an expression that can be evaluated to TRUE or FALSE. The value of the **op** property can be one of the following:
- = (equal to)**—The expression evaluates to TRUE if all input values are equal.
 - > (greater than)**—The expression evaluates to TRUE if each input value is greater than the value that immediately follows it.
 - < (less than)**—The expression evaluates to TRUE if each input value is less than the value that immediately follows it.
 - >= (greater than or equal to)**—The expression evaluates to TRUE if each input value is greater than or equal to the value that immediately follows it.
 - <= (less than or equal to)**—The expression evaluates to TRUE if each input value is less than or equal to the value that immediately follows it.
 - != (not equal to)**—The expression evaluates to TRUE if all input values are different. No two can be equal.
 - and**—The expression evaluates to TRUE if all input values evaluate to TRUE.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

	or —The expression evaluates to TRUE if one or more input values evaluate to TRUE.
input <i>n</i>	Specifies two or more values to be evaluated using the operator specified.
then	Specifies the value to return if the expression evaluates to TRUE.
else	Specifies the value to return if the expression evaluates to FALSE.

Output Property

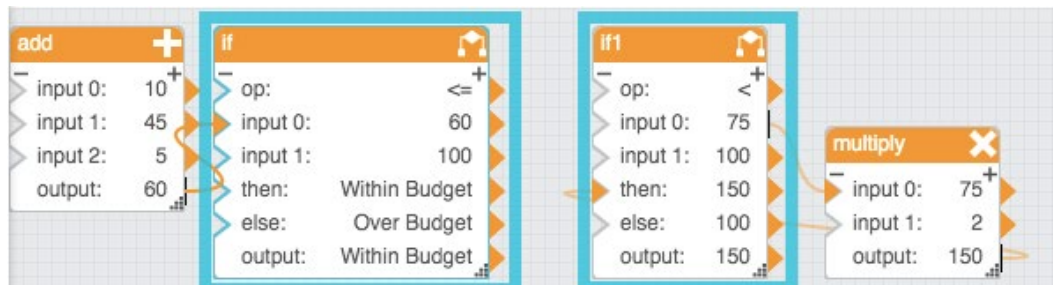
The following property of the If block can give output but cannot take input.

output	Returns either the then value or the else value. The then value is returned if the expression evaluates to TRUE and the else value is returned otherwise.
---------------	---

Examples

Figure 97 demonstrates two examples of the If block. In this image, the leftmost If block returns the string Within Budget if the bound value is less than 100 and returns the string Over Budget otherwise. The rightmost If block returns twice the value of **input 0** if **input 0** is less than 100 and returns 100 otherwise.

Figure 97. If Block



Case

The Case block evaluates a list of conditions and returns the result that corresponds to the first condition that evaluates to TRUE.

Note: An expression will evaluate to FALSE if the data type of the test value cannot be resolved to the data type of the source value.

Input/Output Properties

The following properties of the Case block can take input and give output.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

op (enum)	<p>Specifies the arithmetic, comparison, or text operator to apply to the source value and each test value. It is used in conjunction with input and case n to form an expression that can be evaluated to TRUE or FALSE. The value of the op property can be one of the following:</p> <p>= (equal to)—The expression evaluates to TRUE if input equals case n.</p> <p>> (greater than)—The expression evaluates to TRUE if input is greater than case n.</p> <p>< (less than)—The expression evaluates to TRUE if input is less than case n.</p> <p>>= (greater than or equal to)—The expression evaluates to TRUE if input is greater than or equal to case n.</p> <p>contains— The expression evaluates to TRUE if input contains the sequence of characters in case n.</p> <p>startsWith—The expression evaluates to TRUE if input starts with the sequence of characters in case n.</p> <p>endsWith —The expression evaluates to TRUE if input ends with the sequence of characters in case n.</p> <p>regex—The expression evaluates to TRUE if input satisfies the regular expression in case n.</p>
input	Specifies the source value that is to be compared with each test value.
case n	Specifies a test value to compare with the input.
then n	Specifies the result to return if the expression formed by input , operator , and case n is TRUE.
else	Specifies the result to return if none of the expressions evaluates to TRUE.

Output Property

The following property of the Case block can give output but cannot take input.

output	Returns the then n value for the first case n value that creates a true expression. If none of the expressions is true, output returns the else value.
---------------	--

How to Make a Latch

In a latch, the output changes only if one of the expressions evaluates to TRUE. You can make a latch by binding the **output** property of a Case block to the **else** property.

Kinetic - Edge & Fog Fabric Processing Module

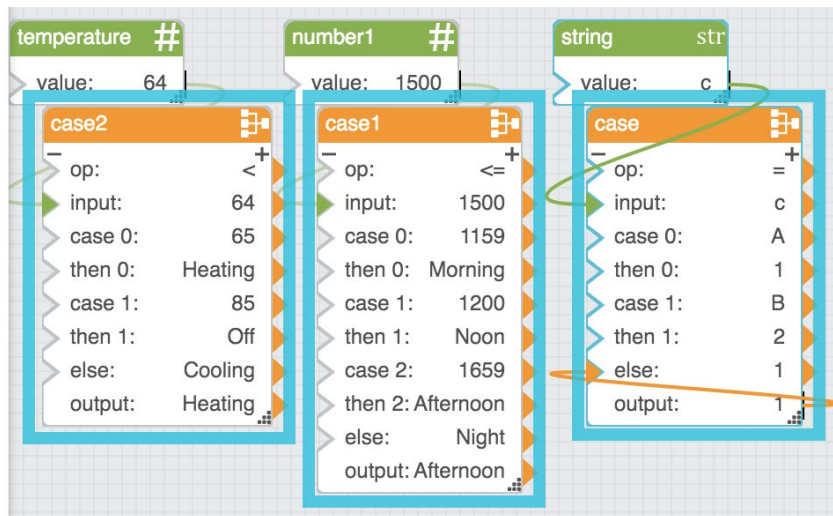
Dataflow Editor User Guide

Dataflow blocks

Examples

Figure 98 demonstrates three examples of the Case block. The leftmost Case block returns the string Heating if **input 0** is below 65, then the string Off if **input 0** is below 85, and then the string Cooling otherwise. The middle Case block returns the name of a time of day that the block determines from a time string. The rightmost Case block is a latch. The latch's output only changes if its input becomes A or B.

Figure 98. Case Block



Hub

The Hub block outputs the most recently updated value from multiple inputs. If events are used as the inputs, the Hub fires **onChange** when any of the events occurs.

Input/Output Property

The following property of the Hub block can take input and give output.

input <i>n</i>	Sets and returns one of the values watched by this hub.
ignoreNull	New parameter with EFM 1.7.0: if set to true, ensures null values will not be passed to the output of the block. For new created blocks the default is "ignoreNull=true", for already existing blocks in existing dataflows it is "ignoreNull=false" for compatibility reasons.

Output Properties

The following properties of the Hub block can give output but cannot take input.

index (number)	Returns the index of the most recently updated input <i>n</i> value. The index is the number in the input's property name, such as the number 2 for input 2 .
-----------------------	---

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

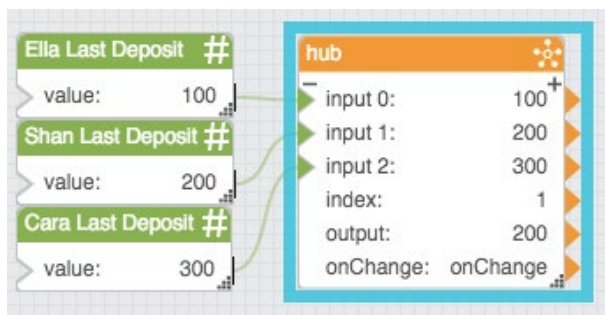
Dataflow blocks

output	Returns the value of the most recently updated input <i>n</i> property.
onChange	Fires when one of the inputs changes or is fired.

Example

Figure 99 shows a Hub block. In this example, Ella deposited at 11:00 a.m., Cara at 1:00 p.m., and Shan at 2:00 p.m. The **output** property is the value of Shan's deposit. Because **input 1** is the most recently updated input, **index** is 1.

Figure 99. Hub Block



Event Gate

The Event Gate block listens for changes to its input properties and fires a new event when all of the input properties have changed.

Optionally, this block can also log the order of the changes it listens for.

A common use of the Event Gate block is to make sure all relevant strings are loaded before data analysis begins.

Note: Make sure to remove any unused input properties, to ensure that the block fires an event when finished.

Input/Output Properties

The following properties of the Event Gate block can take input and give output.

reset (trigger)	Resets the listener and the log.
------------------------	----------------------------------

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

- logOrder** (enum) Specifies whether and how events are logged. Logging ends when all fields have been updated at least once. The value of the **logOrder** property can be one of the following:
- none**—Events are not logged.
 - firstOccurance**—The block logs the order in which each input received its first update. For example, if the order of changes is **input 0**, **input 2**, **input 0**, **input 1**, the logged order is 0,2,1.
 - lastOccurance**—The block logs the order in which each input received its most recent update. For example, if the order of changes is **input 0**, **input 2**, **input 0**, **input 1**, the logged order is 2,0,1.
 - all**—The block logs the order of all input changes, stopping when all fields have been updated at least once. For example, if the order of changes is **input 0**, **input 2**, **input 0**, **input 1**, the logged order is 0,2,0,1.
- input *n*** Specifies one of the values that this block listens to.

Output Properties

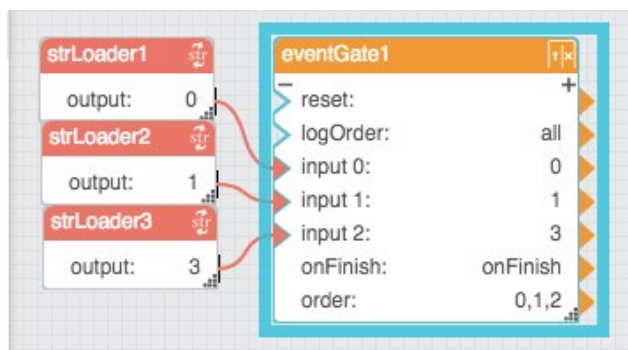
The following properties of the Event Gate block can give output but cannot take input.

- onFinish** (event) Fires when all of the inputs have been updated at least once.
- order** (string) Returns the logged order, as dictated by the **logOrder** property, as a comma-separated list of input indexes.

Example

Figure 100 shows an Event Gate block. In this example, the three strings have been loaded and the event has fired.

Figure 100. Event Gate Block



Script

The Script block holds a custom script and causes the custom script to run.

See also:

- Appendix 1: Dataflow Editor Script
- Appendix 2: Dataflow Editor Syntax
- Appendix 3: Scripts for debugging
- Appendix 4:
- Appendix 5: Operators, Formats, and Functions

Input/Output Properties

The following properties of the Script block can take input and give output.

invoke (trigger)	Causes the script to run.
autoRun (Boolean)	Specifies whether the script runs automatically. TRUE — The script runs every time the script is changed and every time any property of this block is changed. This includes the first time that the script is initialized. FALSE — The script runs only when the invoke property is triggered.
script (string or multi-line text)	Specifies the custom script held by this block.

Additional Properties

You can add more input/output properties to this block and you can specify the new properties' data types and names.

Output Properties

The following properties of the Script block can give output but cannot take input.

output	Returns the output of the custom script.
print (string)	Returns a string that is used for errors, other notifications, and debugging.

Examples

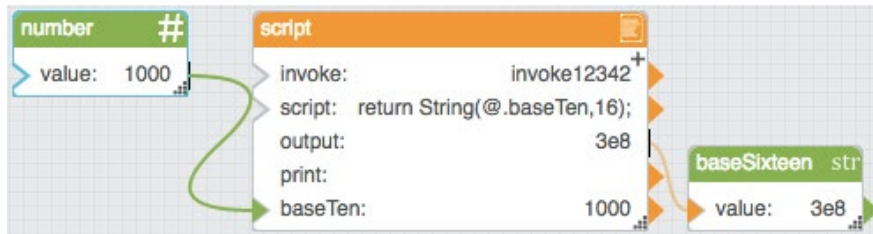
Figure 101 demonstrates an example of the Script block. In this example, the Script block is used to convert a base-ten number to a base-sixteen string.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 101. Script Block



Trace

The Trace block monitors changes to specified properties and logs changes in the browser's JavaScript console. This block is useful for debugging dataflow. It logs changes using the following string: **Trace: <timestamp> : <traceLabel> : <property path> : <msg> : <value>**

The **traceLabel** and **msg** portions of the string appear only if those properties are defined.

Input/Output Properties

The following properties of the Trace block can take input and give output.

tracing (Boolean)	Specifies whether the changes are recorded. TRUE —Changes are logged in the console. FALSE —Changes are not logged.
msg (string)	Specifies a string to be included in the developer console when values for this block are updated.
input <i>n</i>	Specifies one of the properties that this block monitors.

Stop Watch

The Stop Watch block counts up when enabled. An event is fired when the block loops or stops.

Input/Output Properties

The following properties of the Stop Watch block can take input and give output.

enabled (Boolean)	Sets or returns whether the Stop Watch block is currently counting up.
interval (number)	Specifies the duration, in seconds, between updates to the output property. For example, if interval is 0.5, the output property changes every 0.5 seconds.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

step (number)	Specifies the number that is added to the output with each update. For example, if step is 1, then each time the stopwatch output changes, this output increases by 1.
modulo (number)	Specifies the value that signals the stopwatch to loop or stop. If loop is TRUE, then the stopwatch will loop when it reaches the final step before modulo and output is always less than modulo . If loop is FALSE, then the stopwatch will stop when it reaches modulo or the final step before modulo and output is always equal to or less than modulo .
loop (Boolean)	Determines whether the Stop Watch block will start again when it nears modulo . TRUE —The stopwatch restarts as the next step after its output reaches the final step where output is less than modulo . If the difference between the final step and modulo equals the step value, the Stop Watch block restarts at 0. If the difference is less than the step value, the Stop Watch block restarts with a value of that difference. FALSE —The stopwatch output stops changing when it reaches the final step where output is less than or equal to modulo .
reset (trigger)	Causes the stopwatch to restart at 0.
onLoop (event)	Fires when the stopwatch either loops or stops.

Output Property

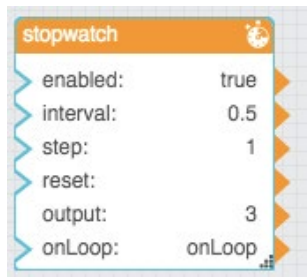
The following property of the Stop Watch block can give output but cannot take input.

output (number)	Returns the current stopwatch value.
------------------------	--------------------------------------

Example

Figure 102 demonstrates a Stop Watch block. In this example, the block counts up by 1 every 0.5 seconds.

Figure 102. Stop Watch Block



Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Delay

The Delay block accepts an input value, waits a predetermined duration, and then returns the same value as output.

Input/Output Properties

The following properties of the Delay block can take input and give output.

input	Holds the value to deliver after a delay.
delay (number)	Sets the length of the delay, in seconds.
interruptMode (enum)	Specifies the behavior of this block in cases where the input changes during a delay. The value of the interruptMode property can be one of the following: <ul style="list-style-type: none"> delay—Every change to the input is sent to output n seconds after it occurs, where n is the delay property. window—A change to the input starts a timer only if the timer is not already running. The latest value is sent when the timer finishes. The window option ensures that updates are sent no more often than once per the duration. wait—A change to input starts or restarts the delay timer. The latest value is sent when the timer finishes. The wait option ensures that updates are sent only when the data stops changing for the duration.

Output Property

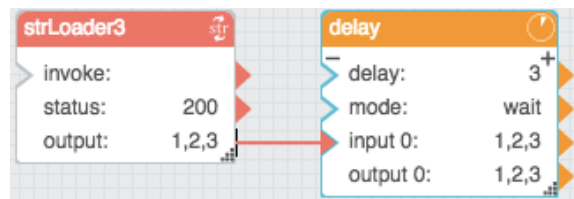
The following property of the Delay block can give output but cannot take input.

output	Returns the input value after the specified delay.
---------------	--

Examples

Figure 103 demonstrates an example of the Delay block. In this example, the Delay block returns the output of the String Loader block only after this property remains unchanged for three seconds.

Figure 103. Delay Block



State

The State block activates and deactivates the defined state upon invoke and revert triggers. A state is a set of property values.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Input/Output Properties

The following properties of the State block can take input and give output.

invoke (trigger)	Activates the state.
revert (trigger)	Deactivates the state.
enabled (Boolean)	Determines whether the Set State block is enabled.
	TRUE —The state is activated and deactivated when the block is invoked.
	FALSE —The state is not activated or deactivated.
duration (number)	
durationBack (number)	
path <i>n</i> (string)	Defines the object whose properties change as part of the state.
change <i>n</i>	Defines the change that happens to the object at path <i>n</i> . To edit the change, click the Imported Layer icon and then edit the properties in the pop-up.

Output Properties

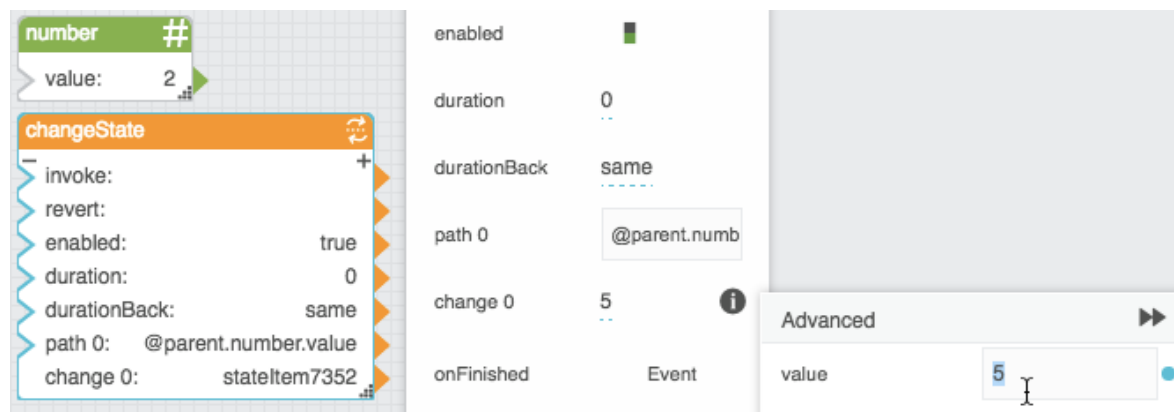
The following properties of the State block can give output but cannot take input.

onFinished (event)	Fires when the activation of the state is complete.
onReverted (event)	Fires when the deactivation or reversion of the state is complete.

Example

Figure 104 shows an example of the State block. In this example, when the State block is invoked, the value property of the Number block will change from two to five.

Figure 104. State Block before Invocation



Catch Error

The Catch Error block returns an error whenever an error is encountered by any block that is assigned to this block's error group.

Input/Output Property

The following property of the Catch Error block can take input and give output.

errorGroup (string) Determines the string that identifies the error group. The **errorGroup** property of the Catch Error block must be identical to the **errorGroup** property of all blocks that this Catch Error block monitors.

Output Property

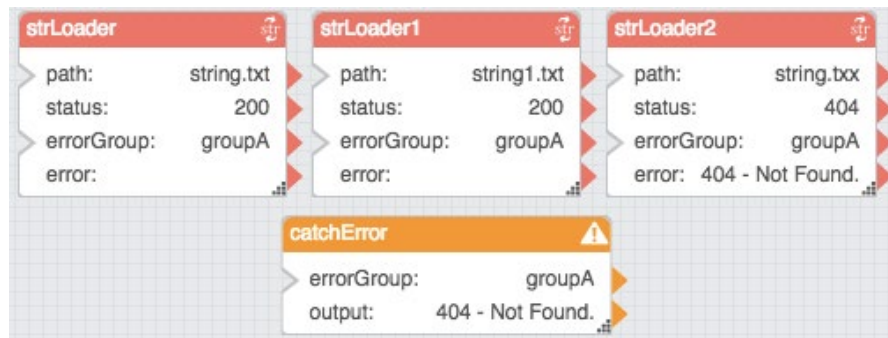
The following property of the Catch Error block can give output but cannot take input.

output (string) Returns the most recent error encountered by any of the blocks in the error group.

Example

Figure 105 demonstrates an example of the Catch Error block. In this example, when the rightmost String Loader block encounters an error, the Catch Error block returns the same error.

Figure 105. Catch Error Block



Make Object

The Make Object block takes input key/value pairs and returns a JSON object.

Input/Output Properties

The Make Object block can have a variable number of properties, for which you specify names. Each of these properties can take input and give output.

Output Property

The following property of the Make Object block can give output but cannot take input.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

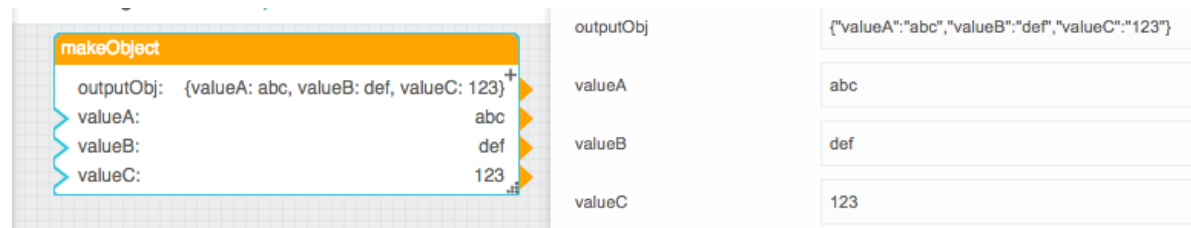
Dataflow blocks

outputObj (JSON object) Returns a JSON object that comprises the input key/value pairs.

Example

Figure 106 shows a Make Object block. In this example, the three input pairs are contained in the output JSON object.

Figure 106. Make Object Block



Repeater

The Repeater block returns a table that, for each row in an input table, lists the parameter values of an input dataflow symbol. See Dataflow symbols and dataflow repeaters.

Input/Output Properties

The following properties of the Repeater block can take input and give output.

- symbol** (string) Specifies the name of the dataflow symbol to repeat.
- data** (table) Specifies the table that determines the number of symbol instances and the properties of each symbol instance.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

mode (enum)	<p>Determines how the repeater handles data changes that happen during and after the rendering of the repeater.</p> <p>normal—A renderer is created for each input row. The repeater's output table is updated at runtime. Each row can be updated multiple times.</p> <p>threaded—A limited number of renderers are created, based on the thread property value. The symbol's onDone trigger property causes a renderer to stop processing. The onDone property must be added manually and will work automatically if it is a trigger property named onDone. When a renderer stops, the symbol is used on a new input row, so the previous output row is not updated further. The repeater's output table is updated only after all input rows are done. If the input table changes before the output is finished, the new input will be in a pending state. The new processing will start after the first rendering is complete. If the input changes multiple times before it can be processed, only the last input will be in a pending state, and others are skipped.</p> <p>stream—As with threaded mode, a limited number of renderers are created and the onDone property, which must be added manually, stops a renderer. Also similarly to threaded, when a renderer stops, its previous rows can't be updated. In contrast to threaded, as soon as any row is finished, it is put in the output table. This means that if multiple renderers finish in the same frame, the output table will have more than one row for some input row. The output order is not always the same as the input order. If the input order is changed, the rows are put into a queue. If the input is changed multiple times, all data is merged into the same queue. The queue has a limited size defined by the streamQueue property. If streamQueue is 0, the queue has unlimited size. Use stream mode with Load Value blocks where qos is 1 or 3.</p>
thread	Determines the number of renderers created for a repeater in threaded mode or stream mode.
streamQueue	Determines the size of the queue for a repeater in stream mode.

Renderer Properties

When **symbol** and **data** are defined, renderer properties appear that match the parameters of the symbol.

To bind input table columns to renderer properties, you can drag and drop the table column headings.

Output Property

The following property of the Repeater block can give output but cannot take input.

output (table)	Returns a table that lists the symbol parameter values for each instance of the symbol.
-----------------------	---

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Example

Figure 107 shows a Repeater block. In this example, the input table's columns have been bound to the sum term properties and the output table includes a product column. For more about this example, see Dataflow Symbol and Repeater .

Figure 107. Repeater Block

The screenshot shows the Dataflow Editor interface. On the left, a 'table' block is connected to the 'data' property of a 'repeater' block. The 'repeater' block's 'symbol' is set to 'multiplySums'. The 'output' property is set to 'Table'. The 'renderer' property is set to 'sum1term1', 'sum1term2', 'sum2term1', 'sum2term2', and 'product'. The 'table' block's data is shown as a table with columns v1, v2, v3, v4 and rows (0, 1, 2). The 'Table' block's data is shown as a table with columns sum1term1, sum1term2, sum2term1, sum2term2, product and rows (0, 1, 2).

row	v1	v2	v3	v4
0	1	4	3	3
1	3	2	2	4
2	2	6	1	2

row	sum1term1	sum1term2	sum2term1	sum2term2	product
0	1	4	3	3	30
1	3	2	2	4	30
2	2	6	1	2	24

String Operations Blocks

These String Operations blocks return information about strings or perform manipulations on strings.

Concatenate

The Concatenate block joins multiple text strings into one string.

Input/Output Properties

The following properties of the Concatenate block can take input and give output.

input *n* (string) Specifies one of the strings to concatenate.

Output Property

The following property of the Concatenate block can give output but cannot take input.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

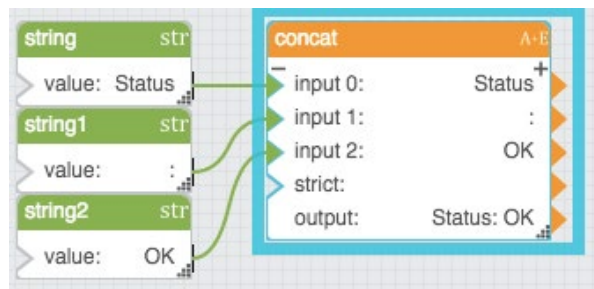
Dataflow blocks

output (string) Returns a string formed by concatenating all of the input values.

Example

Figure 108 demonstrates a Concatenate block. In this example, three strings are concatenated.

Figure 108. Concatenate Block



Left

The Left block returns the specified number of leftmost characters from a string.

Input/Output Properties

The following properties of the Left block can take input and give output.

input (string) Specifies the text string that contains the characters to extract.

length (number) Specifies how many characters to extract.

Output Property

The following property of the Left block can give output but cannot take input.

output (string) Returns the extracted characters.

Special Cases

The following are special cases for the Left block:

- The **length** property must be greater than or equal to zero.
- If **length** is null, it is assumed to be zero.
- If **length** is greater than the length of the text, **output** returns the entire input

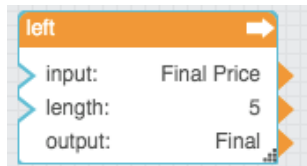
Example

Figure 109 demonstrates a Left block. In this example, the Left block extracts the leftmost 5 characters in the string Final Price and returns the string Final.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 109. Left Block



Length

The Length block returns the number of characters in a string. Spaces are considered characters.

Input/Output Property

The following property of the Length block can take input and give output.

input (string) Specifies the text to find the length for.

Output Property

The following property of the Length block can give output but cannot take input.

output (number) Returns the number of characters in the input string.

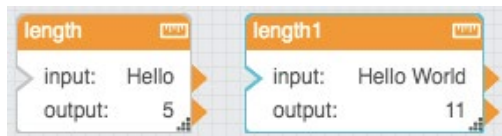
Examples

Figure 110 demonstrates two Length blocks.

The leftmost Length block returns the number of characters in the following string: Hello

The rightmost Length block returns the number of characters in the following string: Hello World

Figure 110. Length Block



Right

The Right block returns the specified number of rightmost characters from a string.

Input/Output Properties

The following properties of the Right block can take input and give output.

input (string) Specifies the text string that contains the characters to extract.

length (number) Specifies how many characters to extract.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Output Property

The following property of the Right block can give output but cannot take input.

output (string) Returns the extracted characters.

Special Cases

The following are special cases for the Right block:

- The **length** property must be greater than or equal to zero.
- If **length** is null, it is assumed to be zero.
- If **length** is greater than the length of the text, **output** returns the entire **input**.

Example

Figure 111 demonstrates a Right block. In this example, the Right block extracts the rightmost 5 characters in the string Final Price and returns the string Price.

Figure 111. Right Block



Substring

The Substring block returns a specified substring from a string.

The block checks for parameters in the following order of importance:

- A start character position and length
- A start character position and end character position
- An end character position and length

Input/Output Properties

The following properties of the Substring block can take input and give output.

input (string) Specifies the text string that contains the characters to extract.

start (number) Determines the position of the first character to extract. A value of 0 indicates the first character in the input.

length (number) Specifies the number of characters to extract.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

end (number) Determines the position of the last character to extract.

Output Property

The following property of the Substring block can give output but cannot take input.

output (string) Returns the substring.

Special Cases

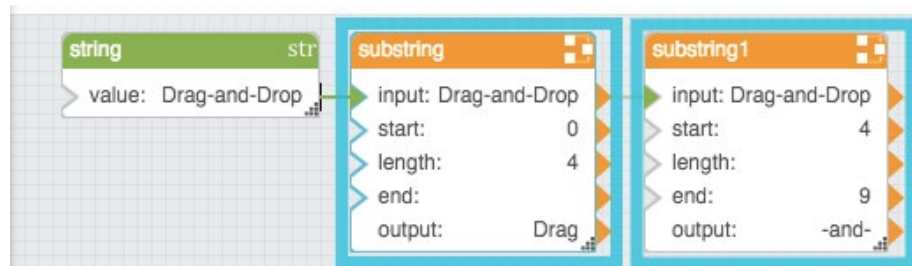
The following are special cases for the Substring block:

- If **start**, **length**, and **end** are null, the full input string is returned.
- If **start** is greater than the length of **input**, the Substring block returns empty text.
- If **start** plus **length** exceeds the length of **input**, the Substring block returns a substring that begins with **start** and ends with the end of the text.

Examples

Figure 112 demonstrates two examples of the Substring block. The leftmost Substring block returns a four-character substring that begins with the character at position 0. The rightmost Substring block returns a substring that begins with the character at position 4 and ends with the character at position 9.

Figure 112. Substring Block



Replace

The Replace block replaces a substring of a text value.

You can replace multiple strings within the text with a single Replace block.

Input/Output Properties

The following properties of the Replace block can take input and give output.

input (string) Specifies the text string in which to replace characters.

regexp (Boolean) Specifies whether to use a regular expression to find strings.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

- find *n*** (string) Specifies the text to find.
- replace *n*** (string) Specifies the text to replace **find *n*** with.

Output Property

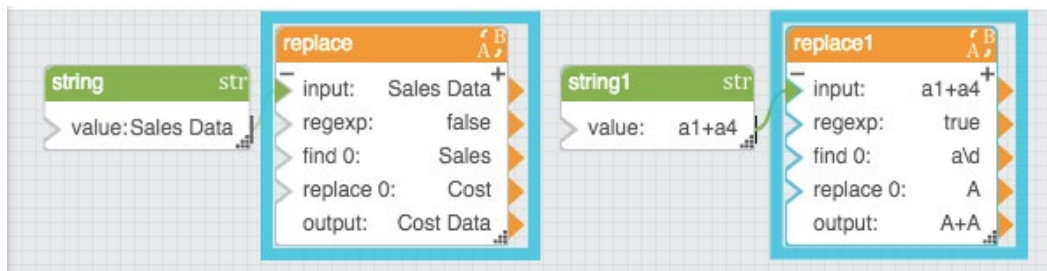
The following property of the Replace block can give output but cannot take input.

- output** (string) Returns the string with all replacements.

Examples

Figure 113 shows two examples of the Replace block. In the leftmost Replace block, the string Sales is replaced with the string Cost. In the rightmost Replace block, matches for the regular expression `a\d` are replaced with A.

Figure 113. Replace Block



Trim

The Trim block removes all leading and trailing whitespace from text.

Input/Output Property

The following property of the Trim block can take input and give output.

- input** (string) Specifies the text to remove whitespace from.

Output Property

The following property of the Trim block can give output but cannot take input.

- output** (string) Returns the input text, with any leading or trailing whitespace removed.

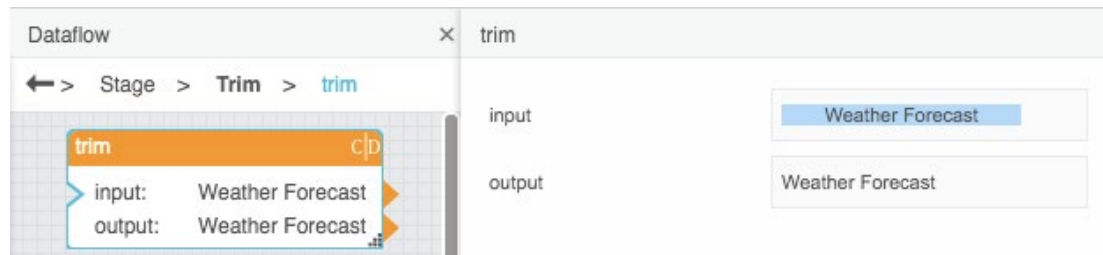
Example

Figure 114 shows an example of a Trim block. In this example, the Trim block removes space characters or Tab characters before and after the string Weather Forecast.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 114. Trim Block



Split

The Split block separates an input string into pieces, as defined by a delimiter, and returns the pieces as rows in an output table.

Input/Output Properties

The following properties of the Split block can take input and give output.

- input** (string) Specifies the string to separate into pieces.
- separator** (string) Specifies the character or string that delimits the pieces.

Output Property

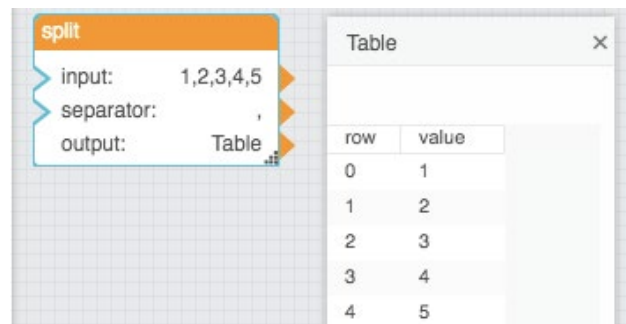
The following property of the Split block can give output but cannot take input.

- output** (table) Returns a table that contains the pieces of the input string as rows.

Example

Figure 115 shows an example of the Split block. In this example, pieces are delimited by commas (,). The output table is also shown in the image.

Figure 115. Split Block



Math Operations Blocks

These Math Operations blocks perform arithmetical operations on numbers.

Error values, and text that cannot be resolved to numbers, cause a **NaN** error result to be returned. The following are accepted and excluded arguments for Math Operations blocks:

- Numbers, logical values, and numbers represented as strings are accepted.
- Error values, and text that cannot be resolved to numbers, are ignored in the calculation.

Add

The Add block adds its arguments.

Input/Output Property

The following property of the Add block can take input and give output.

input *n* (number) Specifies one of the numbers to add.

Output Property

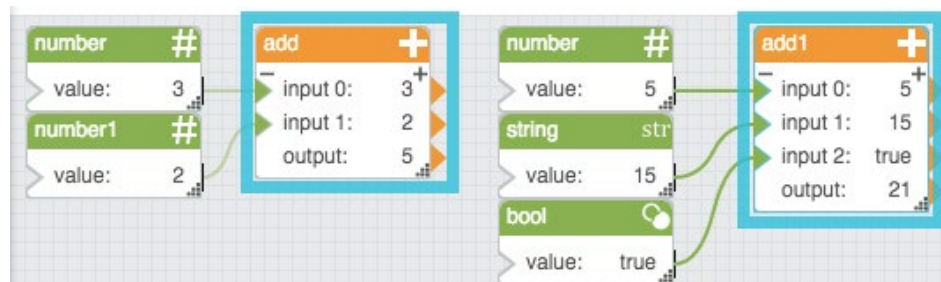
The following property of the Add block can give output but cannot take input.

output (number) Returns the sum of all of the input values.

Examples

Figure 116 shows two examples of the Add block. In the leftmost Add block, two number values are added. In the rightmost Add block, three different data types are used: a number, a string representation of a number, and a Boolean value.

Figure 116. Add Block



Divide

The Divide block divides its arguments.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Division is calculated in order, top to bottom.

Input/Output Properties

The following properties of the Divide block can take input and give output.

input 0 (number) Specifies the initial dividend.

input *n* (number) Specifies a divisor.

Output Property

The following property of the Divide block can give output but cannot take input.

output (number) Returns the result of dividing all of the input values.

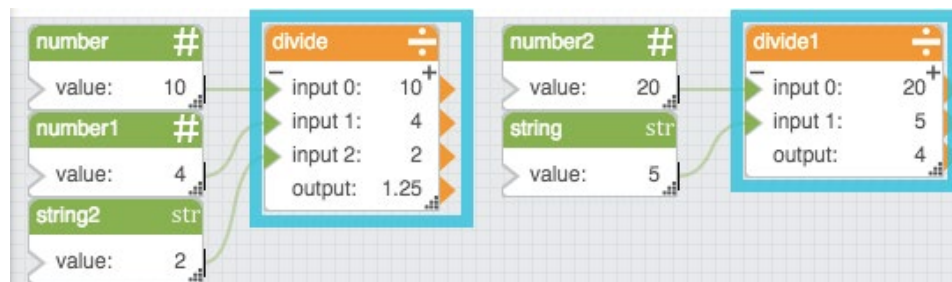
Special Case for the Divide Block

A divisor of zero causes an “Infinity” error result to be returned.

Examples

Figure 117 shows two examples of the Divide block. In the leftmost Divide block, ten is divided first by four and then by two, for an output value of 1.25. In the rightmost Divide block, two different data types are used: a number and a string representation of a number.

Figure 117. Divide Block



Factorial

The Factorial block returns the single factorial of a number. The single factorial of n is equal to $1 * 2 * 3 * \dots * n$.

Input/Output Property

The following property of the Factorial block can take input and give output.

input (number) Specifies the number to find the factorial for.

Output Property

The following property of the Factorial block can give output but cannot take input.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

output (number) Returns the factorial of the input.

Special cases for the Factorial Block

The following are special cases for the Factorial block:

- A negative **input** value causes 1 to be returned.
- If the **input** value is not an integer, it is truncated.
- The factorial of zero is one.
- Error values, and text that cannot be resolved to a number, cause 1 to be returned.

Examples

Figure 118 shows four examples of the Factorial block.

Figure 118. Factorial Block



Logarithm

The Logarithm block calculates the logarithm of a number using a specified base.

Input/Output Properties

The following properties of the Logarithm block can take input and give output.

input (number) Specifies the number to get the logarithm for.

base (number) Specifies the base for the logarithm.

Output Property

The following property of the Logarithm block can give output but cannot take input.

output (number) Returns the logarithm for the specified **input** and **base**.

Special case for the Logarithm Block

A negative **input** value causes a **NaN** error result to be returned.

Examples

Figure 119 shows four examples of the Logarithm block.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 119. Logarithm Block



Modulo

The Modulo block returns the quotient and remainder from division.

Input/Output Properties

The following properties of the Modulo block can take input and give output.

input (number)	Specifies the dividend.
modulo (number)	Specifies the divisor.

Output Properties

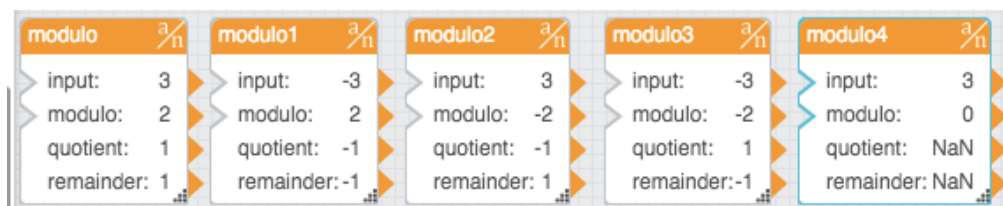
The following properties of the Modulo block can give output but cannot take input.

quotient (number)	Returns the integer result of input divided by modulo .
remainder (number)	Returns the remainder. The remainder property always has the same sign as the input property.

Examples

Figure 120 shows five examples of the Modulo block.

Figure 120. Modulo Block



Multiply

The Multiply block multiplies its arguments.

Input/Output Property

The following property of the Multiply block can take input and give output.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

input n (number) Specifies one of the numbers to multiply

Output Property

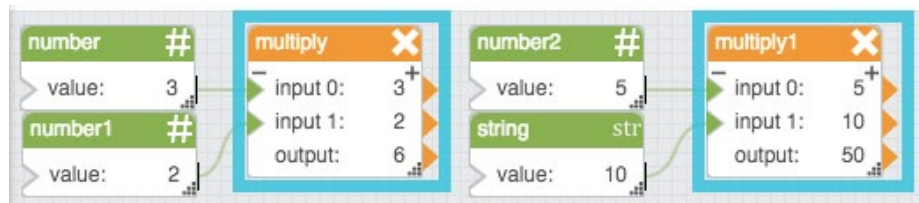
The following property of the Multiply block can give output but cannot take input.

output (number) Returns the result of multiplying all of the input values.

Examples

Figure 121 shows two examples of the Multiply block. In the rightmost Multiply block, two different data types are used: a number and a string representation of a number.

Figure 121. Multiply Block



Power

The Power block returns the result of raising a number to an exponent.

Note: Error values, and text that cannot be resolved to numbers, cause a **NaN** error result to be returned.

Input/Output Properties

The following properties of the Power block can take input and give output.

input (number) Specifies the value to be raised to an exponent.

power (number) Specifies the exponent to raise **input** to.

Output Property

The following property of the Power block can give output but cannot take input.

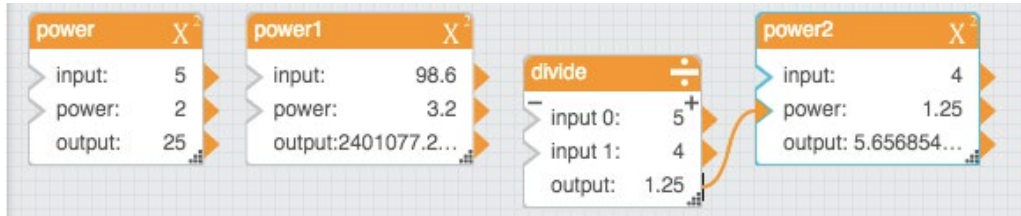
output (number) Returns the result of the calculation.

Examples

Figure 122 shows three examples of the Power block.

Dataflow blocks

Figure 122. Power Block



Root

The Root block returns the n th root of a number. The **input** property must be a non-negative number.

Input/Output Properties

The following properties of the Root block can take input and give output.

- input** (number) Specifies the value to take the root of.
- root** (number) Specifies the root to use. For example, a value of 3 calculates a cubic root.

Output Property

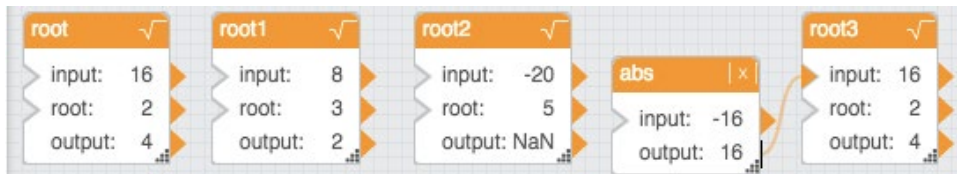
The following property of the Root block can give output but cannot take input.

- output** (number) Returns the result of the calculation.

Examples

Figure 123 shows four examples of the Root block.

Figure 123. Root Block



Scale

The Scale block returns the result of re-scaling a number from one defined set of lower and upper limits to another.

Evaluation of the result is according to the following formula:

- $(\text{input} - \text{inputMin}) * ((\text{outputMax} - \text{outputMin}) / (\text{inputMax} - \text{inputMin})) = \text{output}$

Input/Output Properties

The following properties of the Scale block can take input and give output.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

input (number)	Specifies the number to re-scale.
inputMin (number)	Specifies the minimum of the original scale.
inputMax (number)	Specifies the maximum of the original scale.
outputMin (number)	Specifies the minimum of the new scale.
outputMax (number)	Specifies the maximum of the new scale.
inputOutOfRange (Boolean)	Returns TRUE if the input is outside the range, FALSE otherwise.

Output Property

The following property of the Scale block can give output but cannot take input.

output (number)	Returns the value in the new scale that corresponds to input in the old scale. If input is below the input range, output returns outputMin . If input is above the input range, output returns outputMax .
------------------------	---

Examples

Figure 124 shows four examples of the Scale block.

Figure 124. Scale Block

scale	scale1	scale2	scale3
input: 50	input: -50	input: 51	input: -1
inputMin: 0	inputMin: -100	inputMin: 0	inputMin: 0
inputMax: 100	inputMax: 0	inputMax: 50	inputMax: 50
outputMin: 0	outputMin: 0	outputMin: 0	outputMin: 0
outputMax: 2000	outputMax: 100	outputMax: 100	outputMax: 100
inputOutOfRange:false	inputOutOfRange:false	inputOutOfRange:true	inputOutOfRange:true
output: 1000	output: 50	output: 100	output: 0

Subtract

The Subtract block subtracts its arguments.

Subtraction is calculated in order, top to bottom.

Input/Output Property

The following property of the Subtract block can take input and give output.

input <i>n</i> (number)	Specifies one of the numbers to subtract.
--------------------------------	---

Output Property

The following property of the Subtract block can give output but cannot take input.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

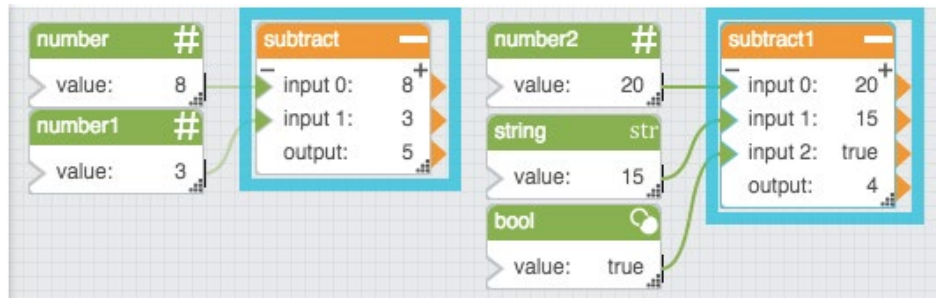
Dataflow blocks

output (number) Returns the result of subtracting all of the input values.

Examples

Figure 125 shows two examples of the Subtract block. In the rightmost Subtract block, two different data types are used: a number and a string representation of a number.

Figure 125. Subtract Block



Number Formatting Blocks

Number Formatting blocks perform formatting operations on numbers.

The following are accepted and excluded arguments for Number Formatting blocks:

- Numbers, logical values, and numbers represented as strings are accepted.
- Error values, and text that cannot be resolved to numbers, cause a **NaN** error result to be returned.

Absolute

The Absolute block returns the absolute value of a number.

Input/Output Property

The following property of the Absolute block can take input and give output.

input (number) Specifies the number for which to calculate absolute value.

Output Property

The following property of the Absolute block can give output but cannot take input.

output (number) Returns the absolute value of **input**.

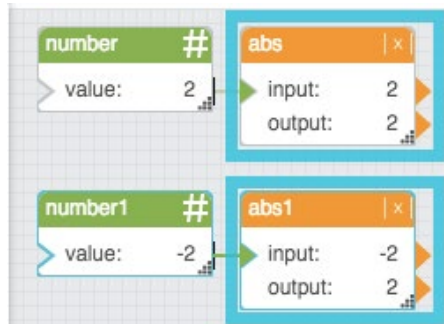
Examples

Figure 126 shows two examples of the Absolute block.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 126. Absolute Block



Bound

The Bound block returns the input number if the input is within the defined minimum and maximum. If the number is above the maximum, this block returns the maximum; if the number is below the minimum, this block returns the minimum.

Input/Output Properties

The following properties of the Bound block can take input and give output.

input (number)	Specifies the number to compare to the range.
min (number)	Specifies the minimum of the range.
max (number)	Specifies the maximum of the range.

Output Properties

The following properties of the Bound block can give output but cannot take input.

inputOutOfRange (Boolean)	Returns TRUE if the input is outside the range, FALSE otherwise.
output (number)	Returns the input number if it is within the range, the min value if input is below the range, or the max value if input is above the range.

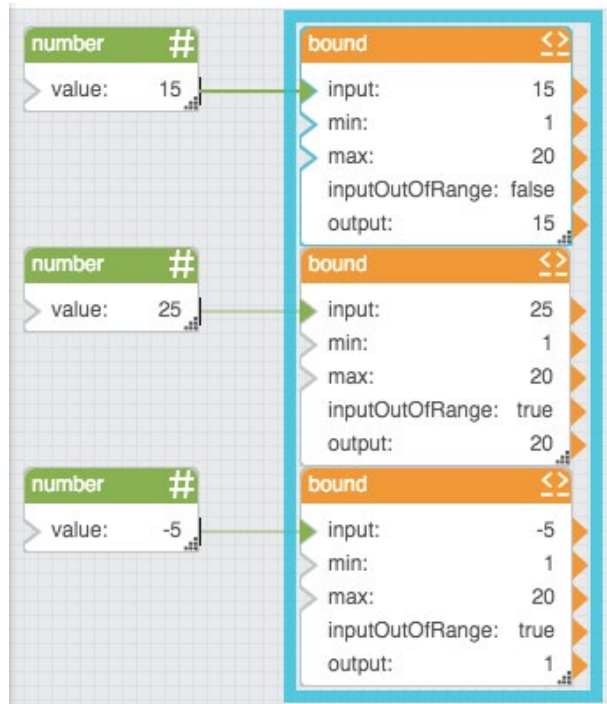
Example

Figure 127 shows three examples of the Bound block. In the topmost Bound block, the input is returned. In the middle Bound block, the input is above the range, so the maximum is returned. In the bottommost Bound block, the input is below the range, so the minimum is returned.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 127. Bound Block



Round

The Round block rounds a number to a specified precision.

Input/Output Properties

The following properties of the Round block can take input and give output.

- | | |
|---------------------------|---|
| input (number) | Specifies any real number that you want to round. |
| precision (number) | Specifies the number of digits after the decimal to round the number to. A value of 0, less than 0, or null rounds the number to the nearest integer. |

Output Property

The following property of the Round block can give output but cannot take input.

- | | |
|------------------------|-----------------------------|
| output (number) | Returns the rounded number. |
|------------------------|-----------------------------|

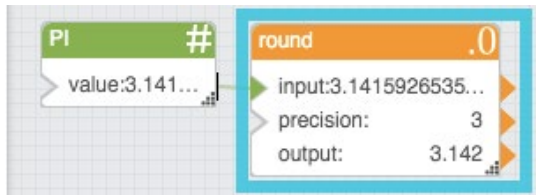
Example

Figure 128 shows an example of the Round block. In this example, pi is rounded to a precision of three digits.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 128. Round Block



Round Up

The Round Up block rounds a number up, away from zero if positive and toward zero if negative.

Input/Output Properties

The following properties of the Round Up block can take input and give output.

input (number)	Specifies any real number that you want rounded up.
precision (number)	Specifies the number of digits after the decimal to round the number to. A value of 0, less than 0, or null rounds the number up to an integer.

Output Property

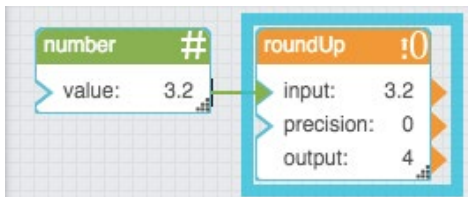
The following property of the Round Up block can give output but cannot take input.

output (number)	Returns the rounded-up number.
------------------------	--------------------------------

Example

Figure 129 shows an example of the Round Up block. In this example, the number 3.2 is rounded up to an integer.

Figure 129. Round Up Block



Round Down

The Round Down block rounds a number down, toward zero if positive and away from zero if negative.

Input/Output Properties

The following properties of the Round Down block can take input and give output.

input (number)	Specifies any real number that you want rounded down.
-----------------------	---

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

precision (number) Specifies the number of digits after the decimal to round the number to. A value of zero, less than zero, or null rounds the number down to an integer.

Output Properties

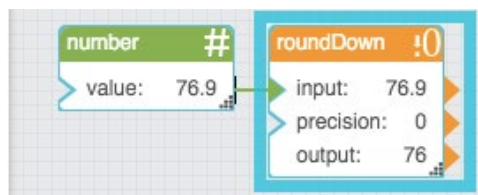
The following properties of the Round Down block can give output but cannot take input.

output (number) Returns the rounded-down number.

Example

Figure 130 shows an example of the Round Down block. In this example, the number 76.9 is rounded down to an integer.

Figure 130. Round Down Block



Format Number

The Format Number block returns a number or string based on defined formatting criteria.

If the input value's number of decimal places is greater than that of the specified format, the Format Number block rounds the value.

Note: If **format** is null, the default formatting is returned. This default formatting includes a thousands separator and all decimal places.

Input/Output Properties

The following properties of the Format Number block can take input and give output.

input (number) Specifies the number to format.

format (number) Specifies a combination of number format patterns to use. For example, you can format the value to be displayed as "1,050.40" with this format string: `#,###.00`

Output Property

The following property of the Format Number block can give output but cannot take input.

output Returns the number or formatted string.

Kinetic - Edge & Fog Fabric Processing Module

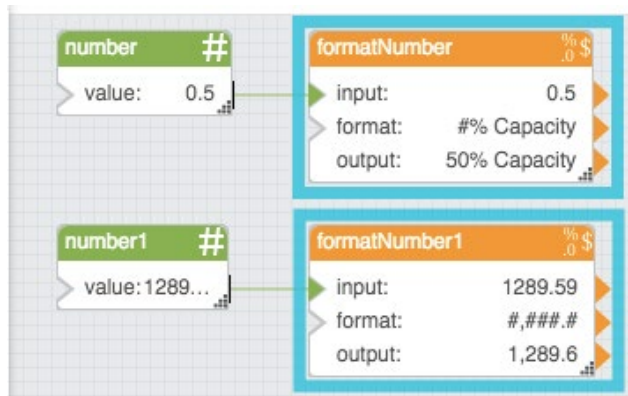
Dataflow Editor User Guide

Dataflow blocks

Examples

Figure 131 shows two examples of the Format Number block. The topmost Format Number block formats a decimal value so that the value appears as a percentage with a label. The bottommost Format Number block rounds a number to the tenths place and includes a comma for a thousands separator.

Figure 131. Format Number Block



Parse Number

The Parse Number block returns the first number that it can parse from a string. If the input string contains more than one numeric value, only the first value is extracted. If the input string does not contain any valid numeric values, a **NaN** error is returned.

Input/Output Property

The following property of the Parse Number block can take input and give output.

input (string) Specifies the string from which to extract a numeric value.

Output Property

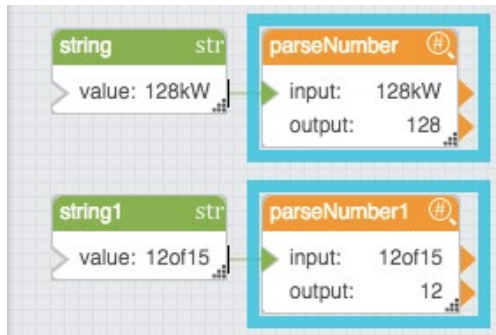
The following property of the Parse Number block can give output but cannot take input.

output (number) Returns the first numeric value parsed from the input string.

Examples

Figure 132 shows two examples of the Parse Number block. The topmost Parse Number block extracts only the number from a string that also includes a unit label. The bottommost Parse Number block extracts the first number from a text string that includes multiple numbers.

Figure 132. Parse Number Block



Statistical Functions Blocks

The Statistical Functions blocks calculate statistical functions on a list of input values. See also: Adding and removing Input/Output properties.

The following are accepted and excluded arguments for Statistical Functions blocks:

- Values of the **input *n*** property can be numbers, arrays, or references that contain numbers.
- Numbers, logical values, and numbers represented as strings are included in calculations.
- Input values are ignored if they are error values, null, or text that cannot be parsed as a number.
- Input values of zero are included in calculations.

Average

The Average block returns the mean of its arguments.

Input/Output Property

The following property of the Average block can take input and give output.

input *n* (number) Specifies one of the values to average.

Output Property

The following property of the Average block can give output but cannot take input.

output (number) Returns the mean of all **input *n*** values.

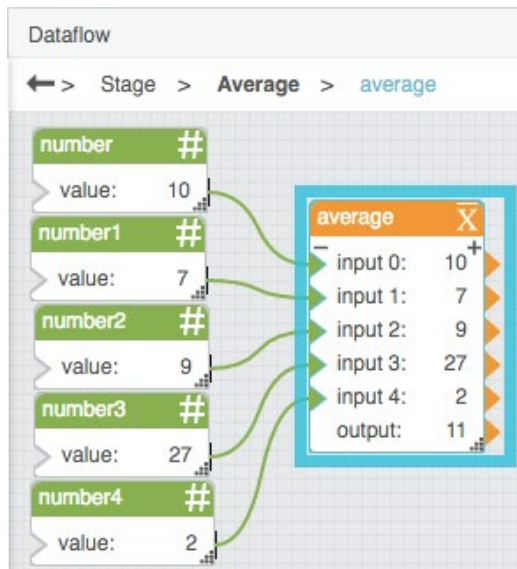
Example

Figure 133 shows an example of the Average block.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 133. Average Block



Maximum

The Maximum block returns the greatest value in a list of arguments.

Note: If none of the **input *n*** values is a number, the Maximum block outputs infinity.

Input/Output Property

The following property of the Maximum block can take input and give output.

input *n* (number) Specifies one of the input values for the statistical function.

Output Property

The following property of the Maximum block can give output but cannot take input.

output (number) Returns the greatest of the input values.

Example

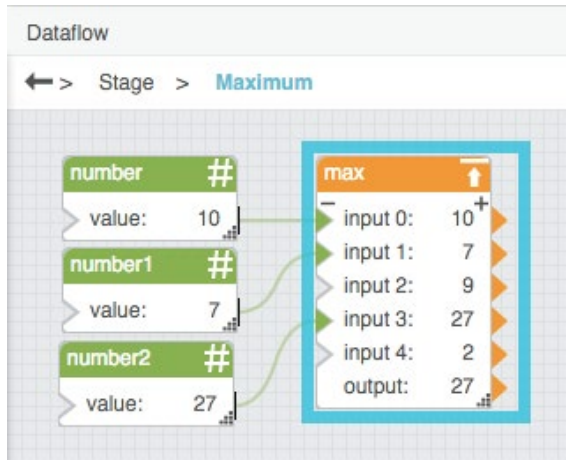
Figure 134 shows an example of the Maximum block.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 134. Maximum Block



Median

The Median block returns the median of its arguments.

Input/Output Property

The following property of the Median block can take input and give output.

input *n* (number) Specifies one of the input values for the statistical function.

Output Property

The following property of the Median block can give output but cannot take input.

output (number) Returns the median of the input values.

Example

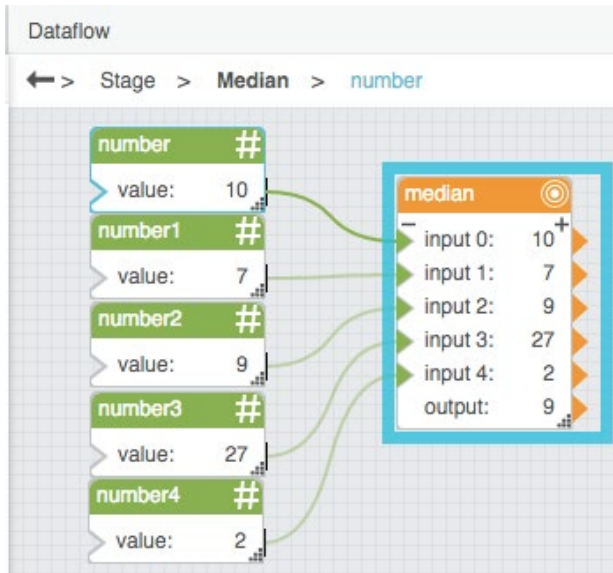
Figure 135 shows an example of the Median block.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 135. Median Block



Minimum

The Minimum block returns the smallest value in a list of arguments.

Note: If none of the **input *n*** values is a number, the block outputs zero.

Input/Output Property

The following property of the Minimum block can take input and give output.

input *n* (number) Specifies one of the input values for the statistical function.

Output Property

The following property of the Minimum block can give output but cannot take input.

output (number) Returns the smallest of the input values.

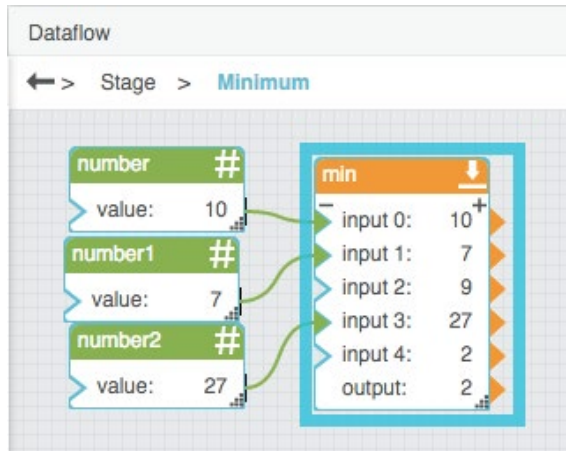
Example

Figure 136 shows an example of the Minimum block.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 136. Minimum Block



Mode

The Mode block returns the most common value in a list of arguments.

Notes:

- If the greatest number of appearances is shared by two or more values, the Mode block returns the value that first reached that number of appearances.
- If none of the **input *n*** values is a number, the block outputs **null**.

Input/Output Property

The following property of the Mode block can take input and give output.

input *n* (number) Specifies one of the input values for the statistical function.

Output Property

The following property of the Mode block can give output but cannot take input.

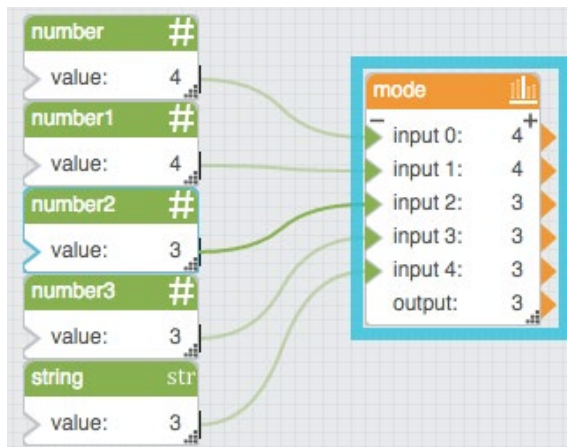
output (number) Returns the mode of all input values.

Example

Figure 137 shows an example of the Mode block.

Dataflow blocks

Figure 137. Mode Block



Standard Deviation

The Standard Deviation block calculates standard deviation based on a list of arguments. The standard deviation is a measure of how widely values are dispersed from the mean.

The Standard Deviation block uses the following expression: $\sqrt{\frac{\sum(x-\bar{x})^2}{(n-1)}}$

In the expression, x is the sample value, \bar{x} is the mean of all x values, and n is the sample size.

Note: If none of the **input n** values is a number, the block outputs **NaN**.

Input/Output Property

The following property of the Standard Deviation block can take input and give output.

input n (number) Defines one of the numbers in the sample.

Output Property

The following property of the Standard Deviation block can give output but cannot take input.

output (number) Returns the standard deviation of the input values.

Example

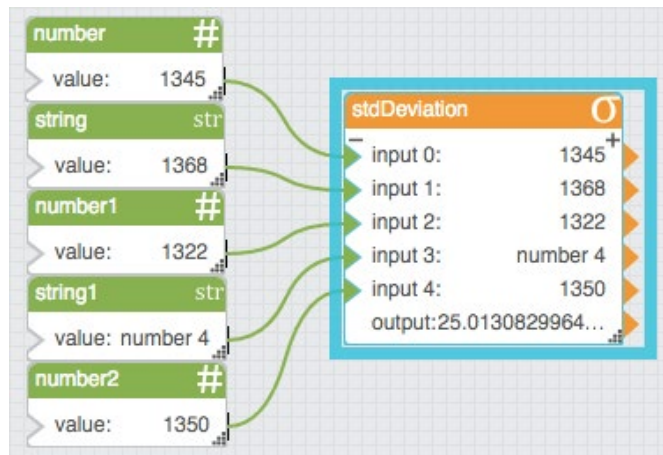
Figure 138 shows an example of the Standard Deviation block.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 138. Standard Deviation Block



Variance

The Variance block calculates the statistical variance of a list of arguments.

The Variance block uses the following equation: $s^2 = \frac{\sum(x-\bar{x})^2}{n}$

In this equation, s^2 is the variance, x is the sample value, \bar{x} is the mean of all x values, and n is the sample size.

Input/Output Property

The following property of the Variance block can take input and give output.

input n (number) Defines one of the numbers in the set.

Output Properties

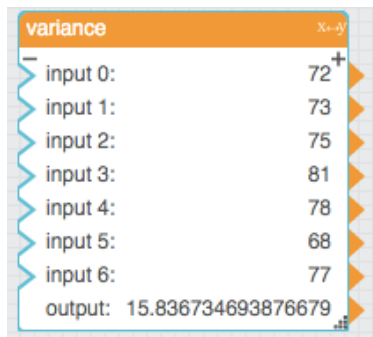
The following properties of the Variance block can give output but cannot take input.

output (number) Returns the variance of the input values.

Example

Figure 139 shows an example of the Variance block.

Figure 139. Variance Block



Trigonometric Functions Blocks

These Trigonometric Functions blocks perform trigonometry operations on an input number and return an output number.

Arc Cosine

The Arc Cosine block returns the arccosine, or inverse cosine, of a number, in degrees. The arccosine of n is the angle for which n is the cosine.

Input/Output Property

The following property of the Arc Cosine block can take input and give output.

input (number) Defines the cosine to get the angle for. Must be between -1 and 1.

Output Property

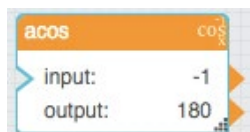
The following property of the Arc Cosine block can give output but cannot take input.

output (number) Returns the arccosine of the input, in degrees.

Example

Figure 140 shows an example of the Arc Cosine block. In this example, the cosine is -1, and the arccosine is 180 degrees.

Figure 140. Arc Cosine Block



Dataflow blocks

Arc Sine

The Arc Sine block returns the arcsine, or inverse sine, of a number, in degrees. The arcsine of n is the angle for which n is the sine.

Input/Output Property

The following property of the Arc Sine block can take input and give output.

input (number) Defines the sine to get the angle for. Must be between -1 and 1.

Output Property

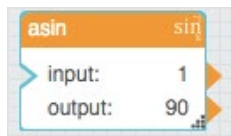
The following property of the Arc Sine block can give output but cannot take input.

output (number) Returns the arcsine of the input, in degrees.

Example

Figure 141 shows an example of the Arc Sine block. In this example, the sine is 1, and the arcsine is 90 degrees.

Figure 141. Arc Sine Block



Arc Tangent

The Arc Tangent block returns the arctangent, or inverse tangent, of a number, in degrees. The arctangent of n is the angle for which n is the tangent.

Input/Output Property

The following property of the Arc Tangent block can take input and give output.

input (number) Defines the tangent to get the angle for.

Output Property

The following property of the Arc Tangent block can give output but cannot take input.

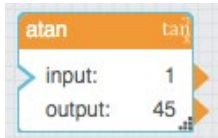
output (number) Returns the arctangent of the input, in degrees.

Example

Figure 142 shows an example of the Arc Tangent block. In this example, the tangent is 1 and the arctangent is 45 degrees.

Dataflow blocks

Figure 142. Arc Tangent Block



Cosine

The Cosine block returns the cosine of the given angle.

Input/Output Property

The following property of the Cosine block can take input and give output.

input (number) Specifies the angle to get the cosine for, in degrees.

Output Property

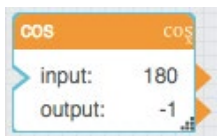
The following property of the Cosine block can give output but cannot take input.

output (number) Returns the cosine of the input.

Example

Figure 143 shows an example of the Cosine block. In this example, the angle is 180 degrees and the cosine is -1.

Figure 143. Cosine Block



Cotangent

The Cotangent block returns the cotangent of an angle, in degrees.

Input/Output Property

The following property of the Cotangent block can take input and give output.

input (number) Specifies the angle to get the cotangent for, in degrees.

Output Property

The following property of the Cotangent block can give output but cannot take input.

output (number) Returns the cotangent of the input.

Kinetic - Edge & Fog Fabric Processing Module

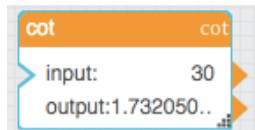
Dataflow Editor User Guide

Dataflow blocks

Example

Figure 144 shows an example of the Cotangent block. In this example, the angle is 30 degrees and the cotangent is 1.732.

Figure 144. Cotangent Block



Degree

The Degree block converts radians to degrees.

Input/Output Property

The following property of the Degree block can take input and give output.

input (number) Specifies the angle in radians, to convert to degrees.

Output Property

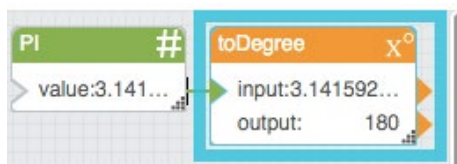
The following property of the Degree block can give output but cannot take input.

output (number) Returns the input value, converted from radians to degrees.

Example

Figure 145 shows an example of the Degree block. In this example, pi is converted from radians to degrees.

Figure 145. Degree Block



Radian

The Radian block converts degrees to radians.

Input/Output Property

The following property of the Radian block can take input and give output.

input (number) Specifies the angle in degrees, to convert to radians.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

Output Property

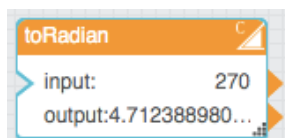
The following property of the Radian block can give output but cannot take input.

output (number) Returns the input value, converted from degrees to radians.

Example

Figure 146 shows an example of the Radian block. In this example, 270 degrees is converted to radians.

Figure 146. Radian Block



Sine

The Sine block returns the sine of the given angle.

Input/Output Property

The following property of the Sine block can take input and give output.

input (number) Specifies the angle to get the sine for, in degrees.

Output Property

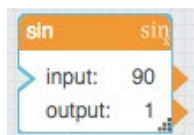
The following property of the Sine block can give output but cannot take input.

output (number) Returns the sine of the input.

Example

Figure 147 shows an example of the Sine block. In this example, the angle is 90 degrees and the sine is one.

Figure 147. Sine Block



Tangent

The Tangent block returns the tangent of the given angle.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

Input/Output Property

The following property of the Tangent block can take input and give output.

input (number) Specifies the angle to get the tangent for, in degrees.

Output Property

The following property of the Tangent block can give output but cannot take input.

output (number) Returns the tangent of the input.

Example

Figure 148 shows an example of the Tangent block. In this example, the angle is 40 and the tangent is 0.839.

Figure 148. Tangent Block

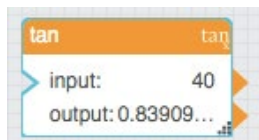


Table Operations Blocks

These Table Operations blocks perform various operations for creating and managing tables.

CSV Parser

The CSV Parser block converts a CSV string into a table.

The CSV parser block is often used to parse a string that has been retrieved using a String Loader block.

Input/Output Properties

The following properties of the CSV Parser block can take input and give output.

input (string) Receives the CSV string. It is often the output from a String Loader block.

withHeader (Boolean) Specifies whether the CSV string includes a header row.

delimiter (character) Specifies the character that separates items in the CSV string. Typically this character is a comma (,).

Output Properties

The following properties of the CSV Parser block can give output but cannot take input.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

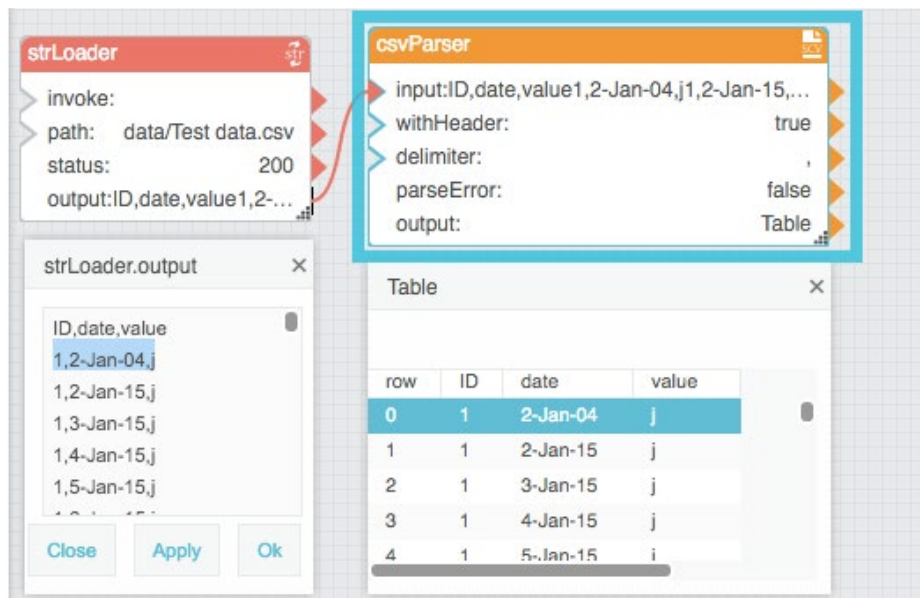
parseError (Boolean)	Indicates whether the block encountered an error. An error might be caused by improperly formatted input.
output (table)	Returns a table parsed from the CSV data.

Example

Figure 149 shows, clockwise from top left:

- A String Loader block
- A CSV Parser block
- The output table of the CSV Parser block
- The CSV file

Figure 149. CSV Parser



CSV Writer

The CSV Writer block takes an input table and returns the table as a CSV string.

Input/Output Properties

The following properties of the CSV Writer block can take input and give output.

input (table)	Receives the table that you want to write as a CSV string.
----------------------	--

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

withHeader (Boolean) Specifies whether the input table contains a header row.

Output Property

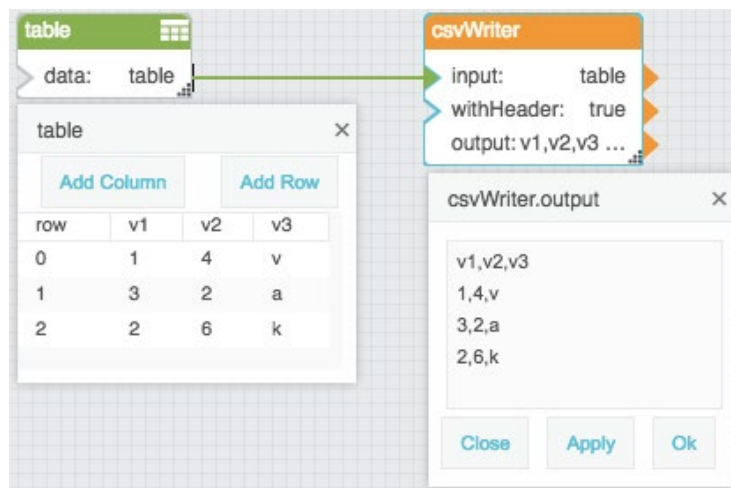
The following property of the CSV Writer block can give output but cannot take input.

output (string) Returns a CSV string of the input table.

Examples

Figure 150 demonstrates an example of the CSV Writer block. In this example, a table with headers is converted to a CSV string. In this example, the input table and output CSV file are displayed in popup windows.

Figure 150. CSV Writer Block



JSON Parser

The JSON Parser block converts a JSON string into a table. The output table can contain other tables within itself. See Viewing the contents of nested tables.

The JSON parser block is often used to parse a string that has been loaded by a String Loader block.

Input/Output Properties

The following properties of the JSON Parser block can take input and give output.

input (string) Receives the JSON string. Sometimes it is the output from a String Loader block.

selector (string) Specifies the location of the data in the JSON string to parse into a table, using the format [**<index>**] [**<identifier>**] [**<index>**] [**<identifier>**]...

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

drillDownDepth (integer)	Specifies how far to drill down into the JSON when extracting data. Data below the drillDownDepth is not included in the output. The drillDownDepth property works only when a drillDownFilter or flatFilter is specified.
drillDownFilter (string)	Specifies what to return for nested tables. Enter * to return all nested tables to the specified drillDownDepth .
flatFilter (string)	Specifies what to return for a flat table. Enter * to return all data to the specified drillDownDepth , in a flat table.

Output Properties

The following properties of the JSON Parser block can give output but cannot take input.

output (table)	Returns a table parsed from the JSON data.
parseError (Boolean)	Indicates whether the block encountered an error. An error might be caused by improperly formatted input.

Example

Figure 151 shows, clockwise from top left:

- A String Loader block
- A JSON Parser block
- The output table of the JSON Parser block
- The JSON file

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 151. JSON Parser Block

The screenshot shows a dataflow diagram with two blocks: **strLoader** and **jsonParser**. The **strLoader** block is on the left, and the **jsonParser** block is on the right, connected by a red arrow. Below the blocks, two windows are open: **strLoader.output** and **Table**.

strLoader.output window content:

```
[
  {
    "ts": "2014-01-01T00:00:00.000",
    "status": "OK",
    "temp": "70"
  },
  {
    "ts": "2014-02-01T00:00:00.000",
    "status": "OK",
    "temp": "70"
  }
]
```

Table window content:

row	ts	status	temp
0	2014-01-01T00:00:00.000	OK	70
1	2014-02-01T00:00:00.000	OK	70
2	2014-03-01T00:00:00.000	OK	70
3	2014-06-01T00:00:00.000	OK	70
4	2014-07-01T00:00:00.000	OK	70
5	2014-08-01T00:00:00.000	OK	70
6	2014-09-01T00:00:00.000	OK	70
7	2014-10-01T00:00:00.000	OK	70

Column Mapping

The Column Mapping block returns a new table that can include specified columns from the input table and can include new columns that contain calculated values. See Appendix 5: Operators, Formats, and Functions.

Input/Output Properties

The following properties of the Column Mapping block can take input and give output.

- input** (table) Receives the input table.
- retainColumns** (Boolean) Specifies whether to include the input table's columns in the output table.
- name *n*** (string) Specifies the new column name.

Dataflow blocks

from <i>n</i> (string)	<p>Specifies the data that will appear in the new column, using JavaScript notation.</p> <p>The following are examples of values for from <i>n</i>:</p> <p>Simple expressions—The value <code>=v1+v2</code> causes this column to hold the sum of the values in the v1 column and the v2 column.</p> <p>DateTime functions—The value <code>=dateFormat(v3, "y-MM-dd")</code> causes this column to reformat and store the value from the v3 column.</p> <p>Number functions—The value <code>=numberFormat(v4, "0.00")</code> causes this column to hold the value in the v4 column, formatted with the specified string.</p> <p>Conditions—The value <code>=(v5 > 1 ? true : false)</code> causes this column to check values in the v5 column. This example returns a TRUE Boolean value if the v5 value is greater than one and a FALSE Boolean value otherwise.</p> <p>Nested conditions—The value <code>=(v6 <= 1 ? false : (v6 > 2 ? false : true))</code> causes this column to check values in the v6 column. If the v6 value is less than or equal to one, this example returns a FALSE Boolean value. Otherwise, this example returns the result of the nested condition.</p> <p>Custom string functions—The value <code>=(function(\$row) {return \$row.length;})(v8)</code> causes this column to perform the string function contained in this from <i>n</i> property and return the result. In this example, the result is the length of the string in v8 for this row.</p>
-------------------------------	--

Output Properties

The following properties of the Column Mapping block can give output but cannot take input.

output (table)	Returns the output table with new columns.
print (string)	Displays error messages and other notifications for debugging.

Example

Figure 152 shows an example of the Column Mapping block. In this example, the new column displays a reformatted date string. The topmost popup window shows the Column Mapping block's input table and the bottommost popup window shows its output table.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 152. Column Mapping Block

The screenshot shows the Dataflow Editor interface. On the left, a `jsonParser` block is connected to a `tableColumnMapping` block. The `tableColumnMapping` block is highlighted with a blue border. Its configuration is as follows:

- input: Table
- retainColumns: true
- name 0: Formatted Date
- from 0: `=dateFormat(ts, "MMMM y")`
- output: Table

Below the blocks, two table views are displayed:

Table 1 (Input):

row	ts	status	temp
0	2014-01-01T00:00:00.000	OK	70
1	2014-02-01T00:00:00.000	OK	70
2	2014-03-01T00:00:00.000	OK	70
3	2014-06-01T00:00:00.000	OK	70
4	2014-07-01T00:00:00.000	OK	70

Table 2 (Output):

row	ts	status	temp	Formatted Date
0	2014-01-01T00:00:00.000	OK	70	January 2014
1	2014-02-01T00:00:00.000	OK	70	February 2014
2	2014-03-01T00:00:00.000	OK	70	March 2014
3	2014-06-01T00:00:00.000	OK	70	June 2014
4	2014-07-01T00:00:00.000	OK	70	July 2014

Sort

The Sort block returns a new table that reorders the input table's rows.

Input/Output Properties

The following properties of the Sort block can take input and give output.

- input** (table) Receives the table that you want to sort.
- column** (string) Specifies the name of the table column to sort by.
- method** (enum) Specifies whether to sort values as strings (alphabetically) or as numbers.
- order** (string) Specifies whether to sort values in ascending or descending order.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Output Property

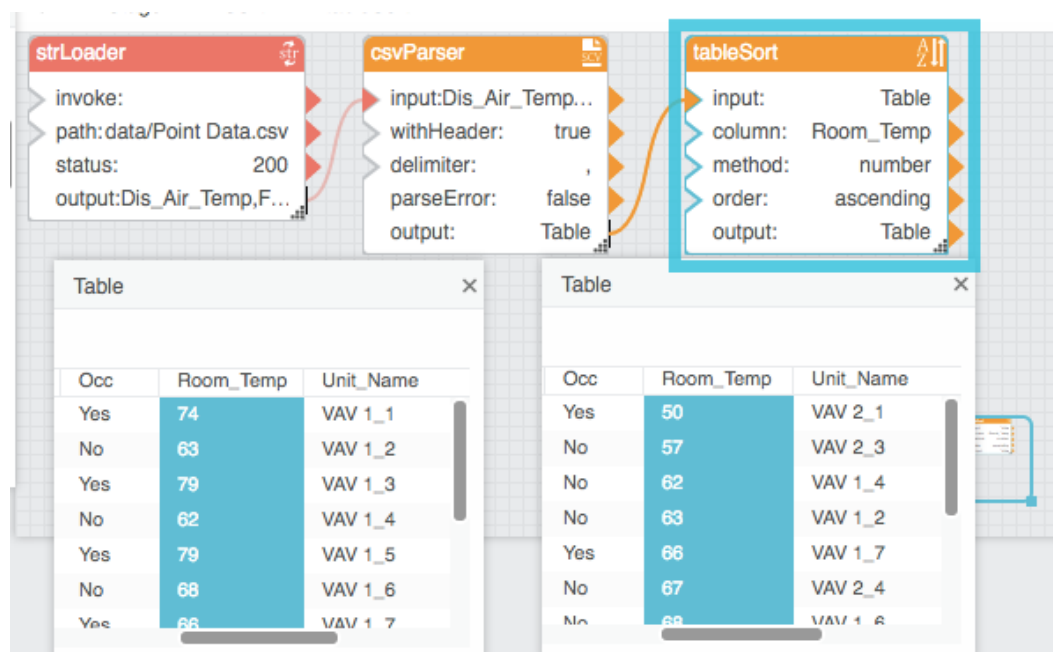
The following property of the Sort block can give output but cannot take input.

output (table) Returns the sorted table.

Example

Figure 153 shows an example of the Sort block. In this example, the Sort block reorders the table rows in ascending order based on the **Room_Temp** column. The leftmost popup window shows the Sort block's input table and the rightmost popup window shows its output table.

Figure 153. Sort Block



Filter

The Filter block returns a new table that contains only the rows from the input table that meet a condition.

Input/Output Properties

The following properties of the Filter block can take input and give output.

input (table) Receives the table that you want to filter.

condition (string) Specifies the condition of the filter. Use JavaScript notation.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

Output Properties

The following properties of the Filter block can give output but cannot take input.

- | | |
|-----------------------|--|
| print (string) | Returns output from the condition field. Use it for debugging. |
| output (table) | Returns the filtered table. |

Examples of the Condition Property

These examples of values for the **condition** property use the following table:

```
row,value
0,string
1,STRING
2,StRiNg
```

Using the table above, the following are example numeric expressions for the **condition** property:

- `row == 0` causes row 0 to be returned.
- `row > 0` causes rows 1 and 2 to be returned.
- `row > -1` causes all rows to be returned.

Using the table above, the following are example string expressions for the **condition** property:

- `String(value) == "string"` causes row 0 to be returned.
- `String(value).indexOf("S") > -1` causes rows 1 and 2 to be returned, because a capital S is included in the string in those rows.
- `String(value).toLowerCase().indexOf("string") > -1` returns all rows, because the strings are converted to lowercase before being tested.

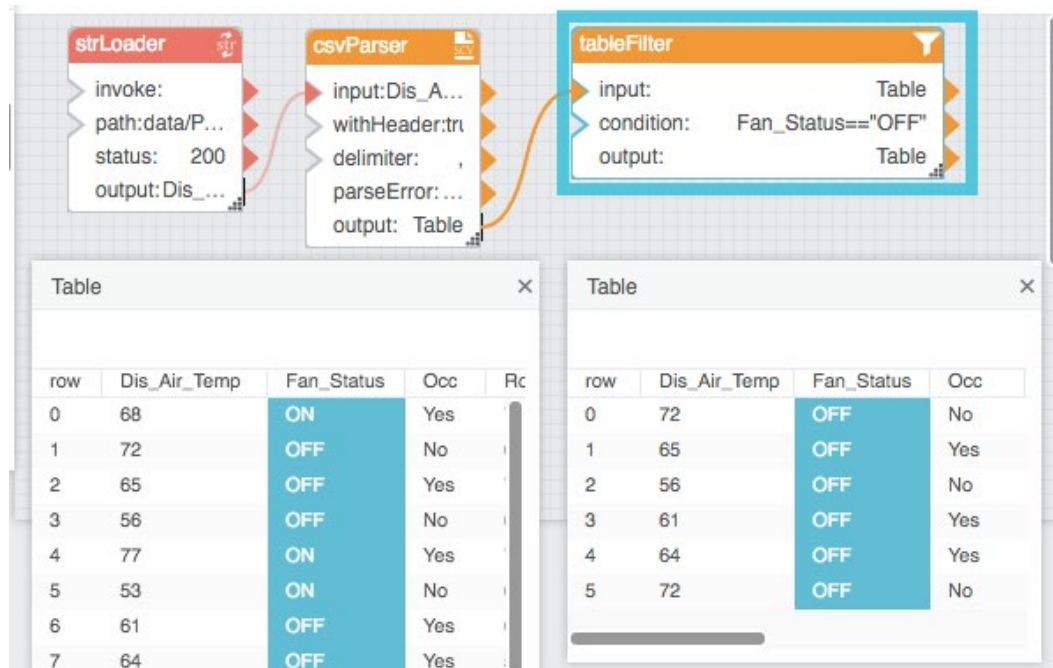
The following example limits a table to include only rows for which the timestamp is on June 14 or June 15, 2017:

```
$thisRow['timestamp'] > '2017-06-14' && $thisRow['timestamp'] <= '2016-06-15T23:59:59'
```

Example of the Filter Block

Figure 154 shows an example of the Filter block. In this example, the table is filtered to contain only rows where the **Fan_Status** column holds the string OFF.

Figure 154. Filter Block



Group By

The Group By block returns a new table that contains one row for each group of rows in the input table. A group is composed of rows that hold an identical value in the specified column.

Input/Output Properties

The following properties of the Group By block can take input and give output.

- input** (table) Receives the table whose values you want to group.
- baseColumn** (string) Specifies the name of the table column to use for creating groups. Rows that hold an identical value in this column will be grouped together.
- nestedData** (Boolean) Specifies whether to keep the original rows and store them in a new column of the output table as nested tables. See Viewing the contents of nested tables.
- includeBlankValue** (Boolean) Specifies whether to create a group for an empty cell in **baseColumn**, if any empty cells exist.
- column *n*** (string) Specifies the name of an input table column to include in the output table.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

method <i>n</i> (enum)	<p>Specifies the value that will be stored in column <i>n</i>. The value of the method <i>n</i> property can be one of the following:</p> <p>first—Returns the first value in the group.</p> <p>last—Returns the last value in the group.</p> <p>average—Returns the average of all number values in the group. Non-number values are treated as null and are not included in the calculation.</p> <p>sum—Returns the sum of all number values in the group. If there are no number values, returns 0.</p> <p>max—Returns the greatest number value in the group. If there are no number values, returns null.</p> <p>min—Returns the smallest number in the group. Non-number values are treated as null and are not included in the calculation.</p> <p>count—Returns the number of values in the group.</p> <p>concat—Returns all values in the group, separated by commas. Duplicate values are included.</p> <p>concatUnique—Returns all unique values in the group, separated by commas. Duplicate values are not included.</p> <p>tableUnion—Returns a table of values.</p>
outColumn <i>n</i> (string)	Specifies the output table column name for the grouped column <i>n</i> values.

Output Property

The following property of the Group By block can give output but cannot take input.

output (table)	Returns the output table.
-----------------------	---------------------------

Example

Figure 155 shows an example of the Group By block. In this example, rows are grouped using the values in the **v1** column. The output table displays a sum of **v2** values and a list of **v3** values for each group.

Figure 155. Group By Block

The screenshot illustrates the configuration and output of the `tableGroupBy` block. The input table is as follows:

row	v1	v2	v3
0	4	1	v
1	2	3	a
2	6	3	k
3			j
4	6	5	h
5	5	6	2
6	5	6	f

The `tableGroupBy` block configuration is:

- input: inputTable
- baseColumn: v1
- nestedData: false
- column 0: v2
- method 0: sum
- outColumn 0: sum of v2
- column 1: v3
- method 1: concat
- outColumn 1: list of v3
- output: Table

The resulting output table is:

row	v1	sum of v2	list of v3
0	4	1	v
1	2	3	a
2	6	8	k,h
3		0	j
4	5	12	2,f

Join

The Join block returns a new table that joins the rows of two input tables.

Note: Make sure to rename columns using the `renameColumns` property if you don't want the columns to be combined.

Input/Output Properties

The following properties of the Join block can take input and give output.

- input1** (table) Specifies the left table for the join operation.
- input2** (table) Specifies the right table for the join operation.
- column1** (string) Specifies the name of the key column from the **input1** table.
- column2** (string) Specifies the name of the key column from the **input2** table.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

join (enum) Determines the method of joining. The value of the **join** property can be one of the following:

Left—All rows from **input1** are included. Rows from **input2** are included and merged when the **column2** value matches a **column1** value. If column names are the same, values in the **input1** table override values in the **input2** table.

Right—All rows from **input2** are included. Rows from **input1** are included and merged when the **column1** value matches a **column2** value. If column names are the same, values in the **input2** table override values in the **input1** table.

Inner—Rows are included and merged only when the values in **column1** and **column2** are the same. If column names are the same, values in the **input1** table override values in the **input2** table.

Full—All rows from both tables are included, and the rows are merged if the values of **column1** and **column2** are the same. If column names are the same, values in the **input1** table override values in the **input2** table.

Union—All rows from both tables are included in a new table.

renameColumns (string) Determines the names in the output table for columns that come from **input2**. Use this syntax:

```
column1:newName
```

```
column2:newName
```

Press Alt + Enter to create line breaks between columns. To edit text in a popup window, click the **Edit in Window** icon in the **renameColumns** field.

Output Property

The following property of the Join block can give output but cannot take input.

output (table) Returns the joined table.

Example

Figure 156 shows an example of the Join block. In this example, a left join operation is performed.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 156. Join Block

The screenshot shows three windows in the Dataflow Editor:

- table**: A table with columns row, ID, value1, value2, value3. Row 2 (ID: C) is highlighted.
- table1**: A table with columns row, ID, value4, value5, value6. Row 0 (ID: C) is highlighted.
- table Join**: A configuration window for a Join block.
 - input1: table
 - input2: table1
 - column1: ID
 - column2: ID
 - join: left
 - renameColumns: value4:a4
 - output: Table
- Table**: The output table resulting from the join. It has columns row, ID, value1, value2, value3, a4, value5, value6. Row 2 (ID: C) is highlighted, showing the joined data.

Page

The Page block returns a specified portion of the input table based on paging parameters.

Input/Output Properties

The following properties of the Page block can take input and give output.

input (table)	Receives the table for which you want to create paging.
pageSize (integer)	Specifies the number of rows per page.
start (integer)	Specifies the row index on which to start the current page. Rows before this index are omitted from the output.
nextPage (trigger)	Advances the output table to the next page.
prevPage (trigger)	Changes the output table to the previous page.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

- firstPage** (trigger) Changes the output table to the first page.
- lastPage** (trigger) Advances the output table to the last page.
- currentPage** (integer) Sets or returns the current page.

Output Properties

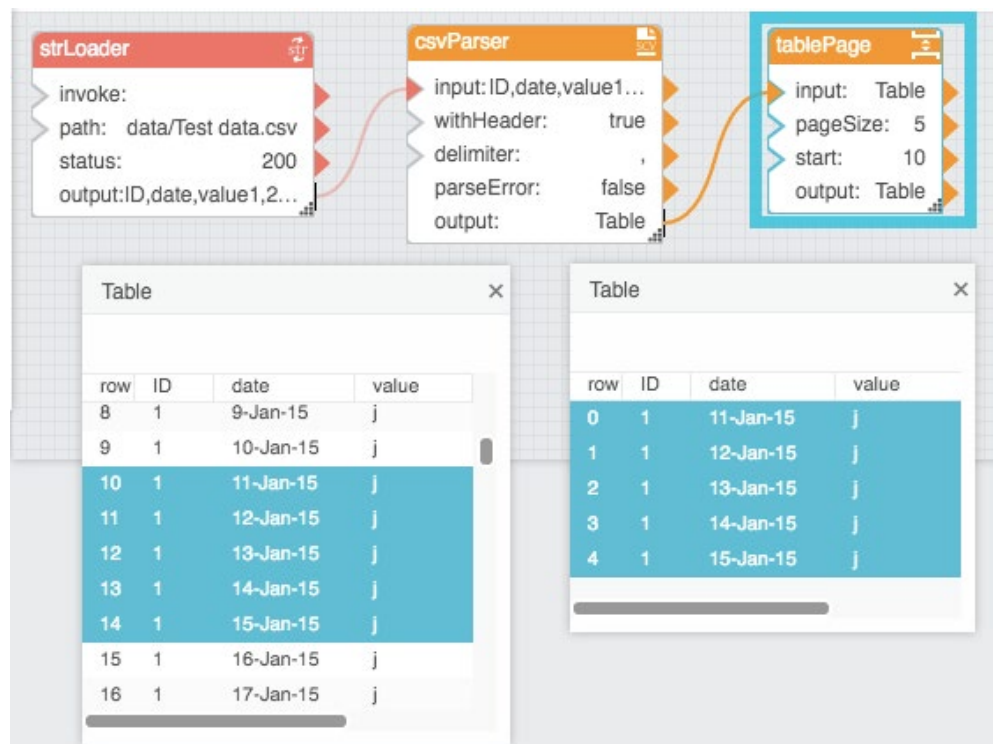
The following properties of the Page block can give output but cannot take input.

- totalPage** (integer) Returns the total number of pages for this table.
- output** (table) Returns the portion of the input table specified by the paging parameters.

Example

Figure 157 shows an example of the Page block. In this example, the output table holds the third page of the input table data; in other words, the output table contains a five-entry page that begins with row 10 of the input data.

Figure 157. Page Block



Rollup

The Rollup block returns a new table that contains one row for each date and time interval in the input table. The output table contains one column for the date and time interval and one column for the value.

Input/Output Properties

The following properties of the Rollup block can take input and give output.

input (table)	Receives the table in which you want to roll up values.
interval (enum)	Specifies the duration of the range to roll up.
dateColumn (string)	Specifies the name of the table column that holds the dates.
valueColumn (string)	Specifies the name of the table column that holds the values to be rolled up.
valueRollup (enum)	Specifies the type of rollup to use. The value of the valueRollup property can be one of the following: First —Returns the first value for the interval. Last —Returns the last value for the interval. Average —Returns the average of all number values for the interval. Non-number values are treated as null and are not included in the calculation. Sum —Returns the sum of all number values for the interval. If there are no number values, returns 0. Max —Returns the greatest number value from the interval. If there are no number values, returns null. Min —Returns the smallest number value for the interval. Non-number values are treated as null and are not included in the calculation. Count —Returns the number of values that exist for the interval. Null values are not included. Concat —Returns all values for the interval, separated by commas. Duplicate values are included. ConcatUnique —Returns all unique values for the interval, separated by commas. Duplicate values are not included.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

roundTime (Boolean)

Specifies whether to force all intervals to start at a round number.

TRUE—Intervals start at a round number. For example, if **interval** is set to **day**, all intervals start at midnight. If **interval** is set to **five minutes**, all intervals start at multiples of five minutes past the hour.

FALSE—The first interval starts at the first timestamp, and intervals are appended accordingly. For example, if **interval** is set to **day** and the first value is 2015/07/19T08:01:19, then all intervals begin at T08:01:19.

Output Property

The following property of the Rollup block can give output but cannot take input.

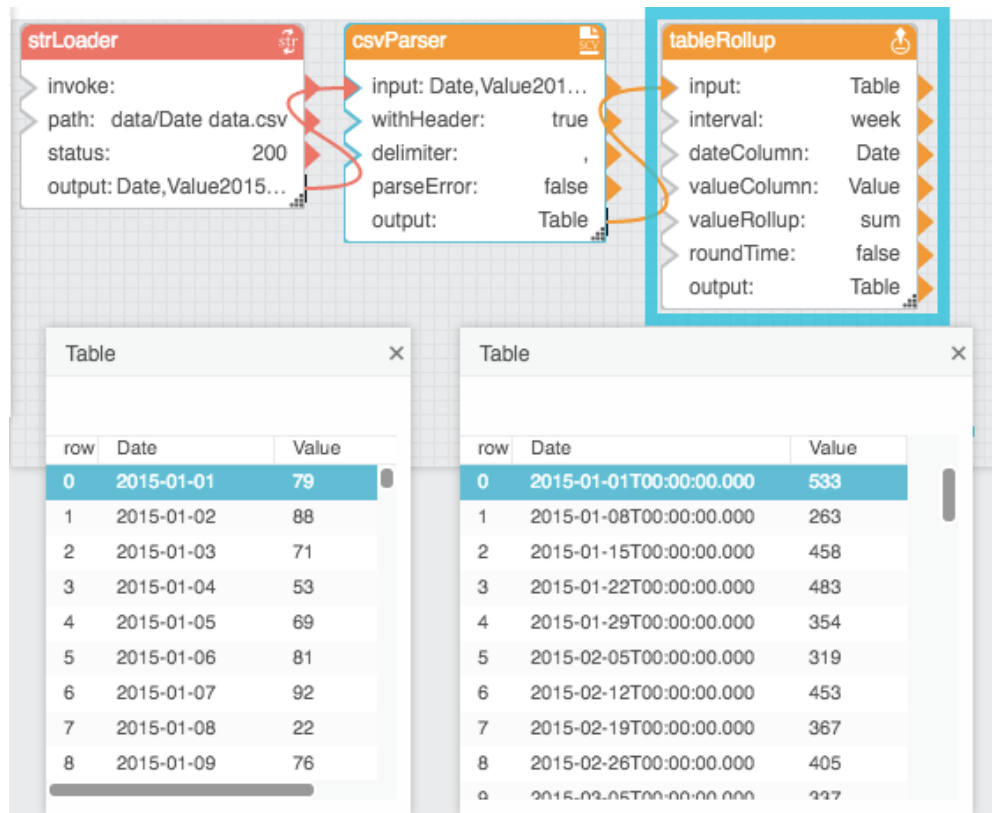
output (table)

Returns the rolled up table.

Example

Figure 158 shows an example of the Rollup block. In this example, the output table contains the sum value for each week.

Figure 158. Rollup Block



Aggregation

The Aggregation block returns a value that reflects the table records in the specified column.

Input/Output Properties

The following properties of the Aggregation block can take input and give output.

input (table)	Receives the table in which you want to aggregate values.
column (string)	Specifies the name of the table column.
method (enum)	Specifies the type of value to return. The value of the method property can be one of the following: First —Returns the first value in the column. Last —Returns the last value in the column. Average —Returns the average of all number values in the column. Non-number values are treated as null and are not included in the calculation. Sum —Returns the sum of all number values in the column. If there are no number values, returns 0. Max —Returns the greatest number value in the column. If there are no number values, returns null. Min —Returns the smallest number value in the column. Non-number values are treated as null and are not included in the calculation. Count —Returns the number of values in the column. Null values are not included. Concat —Returns all values in the column, separated by commas. Duplicate values are included. Concat unique —Returns all unique values in the column, separated by commas. Duplicate values are not included. Table Union —If the input column contains tables, performs a union join on the nested tables and returns the joined table.

Output Property

The following property of the Aggregation block can give output but cannot take input.

output	Returns the output determined by the method property.
---------------	--

Kinetic - Edge & Fog Fabric Processing Module

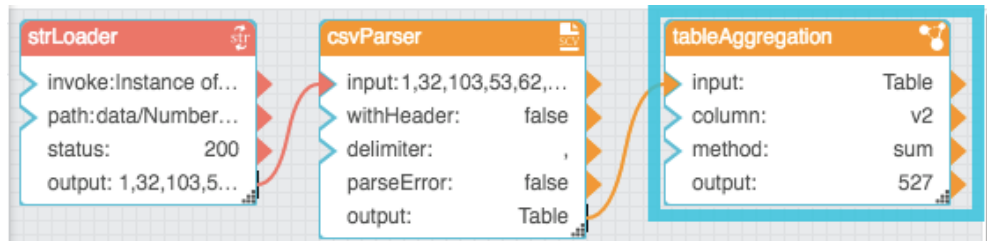
Dataflow Editor User Guide

Dataflow blocks

Example

Figure 159 shows an example of the Aggregation block. In this example, the Aggregation block returns the sum of all values in column **v2** of the input table.

Figure 159. Aggregation Block



Select Rows

The Select Rows block returns a table that contains only the specified rows from the input table.

Input/Output Properties

The following properties of the Select Rows block can take input and give output.

- input** (table) Receives the table from which you want to select rows.
- indexes** (string) Specifies the rows in the input table to select, as comma-separated values.

Output Property

The following property of the Select Rows block can give output but cannot take input.

- output** (table) Returns a new table. It contains only the rows at the specified indexes.

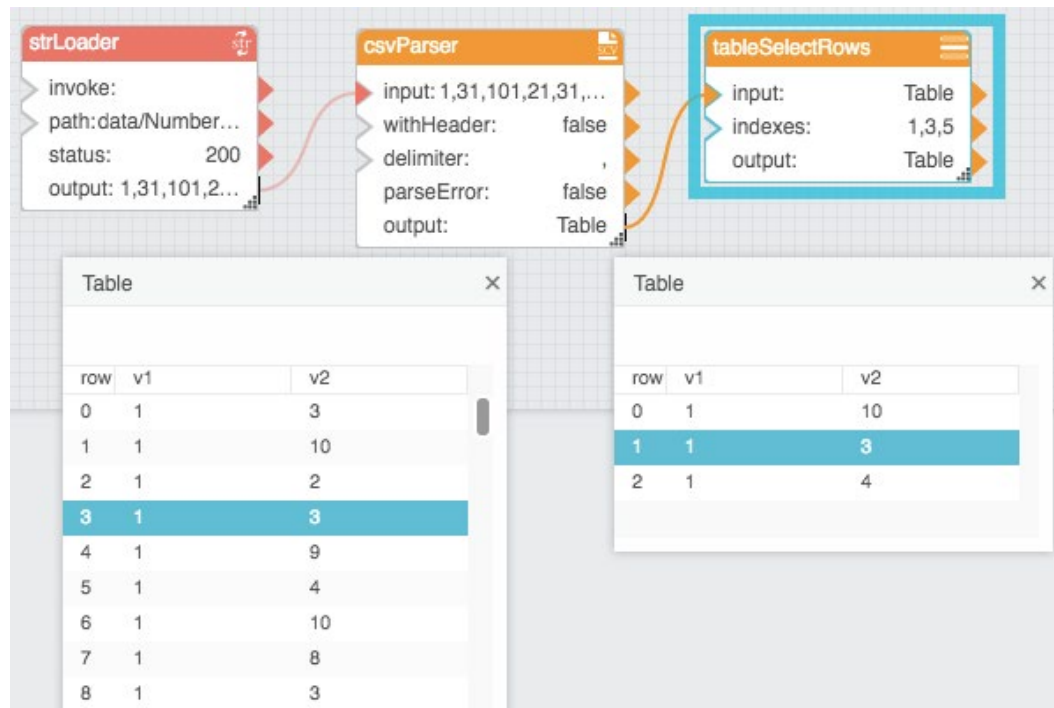
Example

Figure 160 shows an example of the Select Rows block. In this example, the output table contains only rows 1, 3, and 5 from the input table.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 160. Select Rows Block



Select Columns

The Select Columns block returns a table that contains only the defined column names from the input table.

Input/Output Properties

The following properties of the Select Columns block can take input and give output.

- input** (table) Receives the table from which you want to select rows.
- columns** (string) Specifies the columns in the input table to select, as comma-separated column names.

Output Property

The following property of the Select Columns block can give output but cannot take input.

- output** (table) Returns a new table that contains only the specified columns.

Example

Figure 161 demonstrates an example of the Select Columns block. In this example, two columns from the input table are selected.

Figure 161. Select Columns Block



Get Columns

The Get Columns block returns a table that contains the names and data types of the input table's columns.

Note: All data types are string if the input table is parsed from a string. See CSV Parser and JSON Parser.

Input/Output Property

The following property of the Get Columns block can take input and give output.

input (table) Receives the table from which you want to get column names and data types.

Output Property

The following property of the Get Columns block can give output but cannot take input.

output (table) Returns the output table. It contains a row for each column in the input table.

Example

Figure 162 demonstrates an example of the Get Columns block. In this example, an input table with four columns yields an output table with four rows.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 162. Get Columns Block

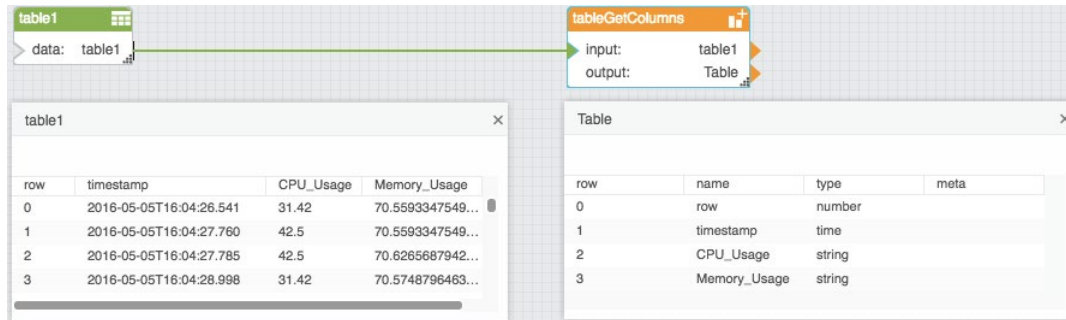


Table Row Cells

The Table Row Cells block returns values from the specified cells in a certain row. In addition, the Table Row Cells block can also write to the input table, if the input table is a Table block.

Caution: In some cases, the Table Row Cells block writes to its input table.

Input/Output Properties

The following properties of the Table Row Cells block can take input and give output.

table (table)	Receives the table from which you want to get values.
row (integer)	Specifies a row index in the input table.
column <i>n</i> (string)	Specifies a column name in the input table.
value <i>n</i>	Sets or returns the cell value at the specified row and column <i>n</i> . Updates to the value <i>n</i> property write to the table if the table is in a Table block. For other data sources, including other block properties that hold the table data type, changing the value does nothing.

Example

Figure 163 demonstrates an example of the Table Row Cells block. In this example, two cell values are returned: the value in row one, column **ID**, and the value in row one, column **status**.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 163. Table Row Cells Block



Add Row

The Add Row block adds a row to the input table. When this block's trigger is invoked, a new row is created in the input table. The new row contains the values specified by this block's properties. The input table must be a table that can be written to, such as the value stored by a Table block.

Caution: The Add Row block writes to its input table.

Notes:

- This block can add values only to existing columns.
- If a value is not specified for an existing column, then the value for that column in the new row is null.

Input/Output Properties

The following properties of the Add Row block can take input and give output.

invoke (trigger)	Causes the row to be added.
enabled (Boolean)	Determines whether the block is enabled.
	TRUE —The row is added when the trigger is invoked.
	FALSE —The row is not added.
table (table)	Specifies the input table that will receive the new row.
column <i>n</i> (string)	Specifies the name of a column in the input table.
value <i>n</i>	Specifies the value to insert in column <i>n</i> in the new row.

Examples

Figure 164 and Figure 165 demonstrate an example of the Add Row block. Figure 164 shows the block and table before invocation and Figure 165 shows the block and table after invocation. In this example, a row with four values is added to a table.

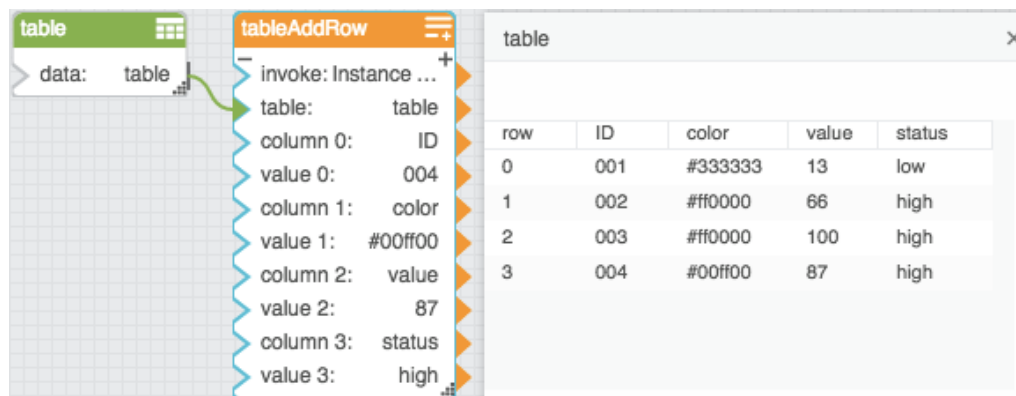
Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 164. Add Row Block before Invocation



Figure 165. Add Row Block after Invocation



Remove Rows

The Remove Rows block deletes rows from the input table. When the block's trigger is invoked, rows that meet the specified criteria are deleted from the input table. The input table must be a table that can be written to, such as the value stored by a Table block.

Caution: The Remove Rows block writes to the input table.

Input/Output Properties

The following properties of the Remove Rows block can take input and give output.

- invoke** (trigger) Causes the rows to be removed.
- enabled** (Boolean) Determines whether the block is enabled.
 - TRUE**—The rows are removed when the trigger is invoked.
 - FALSE**—The rows are not removed.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

- table** (table) Receives the input table from which some rows will be removed.
- condition** (string) Specifies the expression that determines which rows are removed. For example, to remove only the first row from the table, enter `row==0`.

Output Property

The following property of the Remove Rows block can give output but cannot take input.

- print** (string) Returns a string that is used for errors, other notifications, and debugging.

Example

Figure 166 and Figure 167 demonstrate an example of the Remove Rows block. Figure 166 shows the block and table before invocation and Figure 167 shows the block and table after invocation. In this example, the rows where the **status** column holds “medium” are deleted when the block is invoked.

Figure 166. Remove Rows Block before Invocation

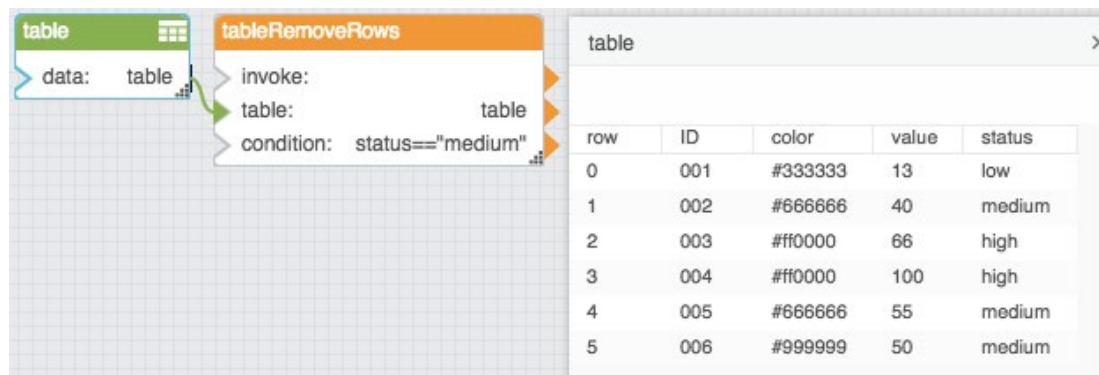
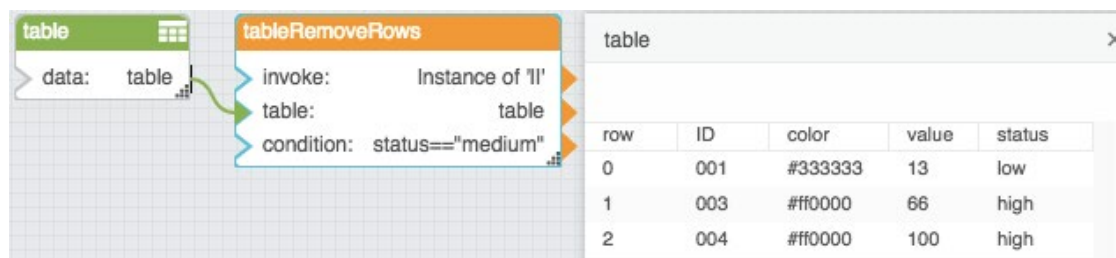


Figure 167. Remove Rows Block after Invocation



Edit Rows

The Edit Rows block replaces values in the input table. When the block’s trigger is invoked, rows that meet the specified criteria receive the specified value replacements. The input table must be a table that can be written to, such as the value stored by a Table block.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Caution: The Edit Rows block writes to its input table.

Input/Output Properties

The following properties of the Edit Rows block can take input and give output.

invoke (trigger)	Causes the values to be replaced.
enabled (Boolean)	Determines whether the block is enabled.
	TRUE —The values are replaced when the trigger is invoked.
	FALSE —The values are not replaced.
table (table)	Receives the input table in which some values will be replaced.
condition (string)	Specifies the expression that determines which rows should be edited. For example, to edit only the first row in the table, enter <code>row==0</code> .
column <i>n</i> (string)	Specifies the name of a column in the input table.
value <i>n</i>	Specifies the value to insert in column <i>n</i> in the input table.

Output Property

The following property of the Edit Rows block can give output but cannot take input.

print (string)	Returns a string that is used for errors, other notifications, and debugging.
-----------------------	---

Example

Figure 168 and Figure 169 demonstrate an example of the Edit Rows block. Figure 168 shows the table and block before invocation and Figure 169 shows the table and block after invocation. In this example, the values in the **color** column are replaced for rows where the value in the **status** column is “high.”

Figure 168. Edit Rows Block before Invocation

row	ID	color	value	status
0	001	#333333	13	low
1	002	#666666	40	medium
2	003	#999999	66	high
3	004	#000000	100	high

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

Figure 169. Edit Rows Block after Invocation



Transpose

The Transpose block returns a new table that transposes the input table, so that the columns in the input table are rows in the output table. Optionally, you can set input columns to omit from the output. You can also set an input column to be the header row of the output table.

Input/Output Properties

The following properties of the Transpose block can take input and give output.

input (table)	Receives the table that you want to transpose.
ignoreColumns (string)	Specifies the names of columns in the input table to ignore when transposing, as comma-separated values.
headerColumn (string)	Specifies the column from the input table to use as headers in the output table.
includeHeaders (Boolean)	Specifies whether the header row from the input table is included as a column in the output table.
headerPrefix (string)	Specifies the text string to include before the column number, for each column header in the output table. The headerPrefix property works only when the headerColumn property is null.

Output Property

The following property of the Transpose block can give output but cannot take input.

output (table)	Returns the transposed table.
-----------------------	-------------------------------

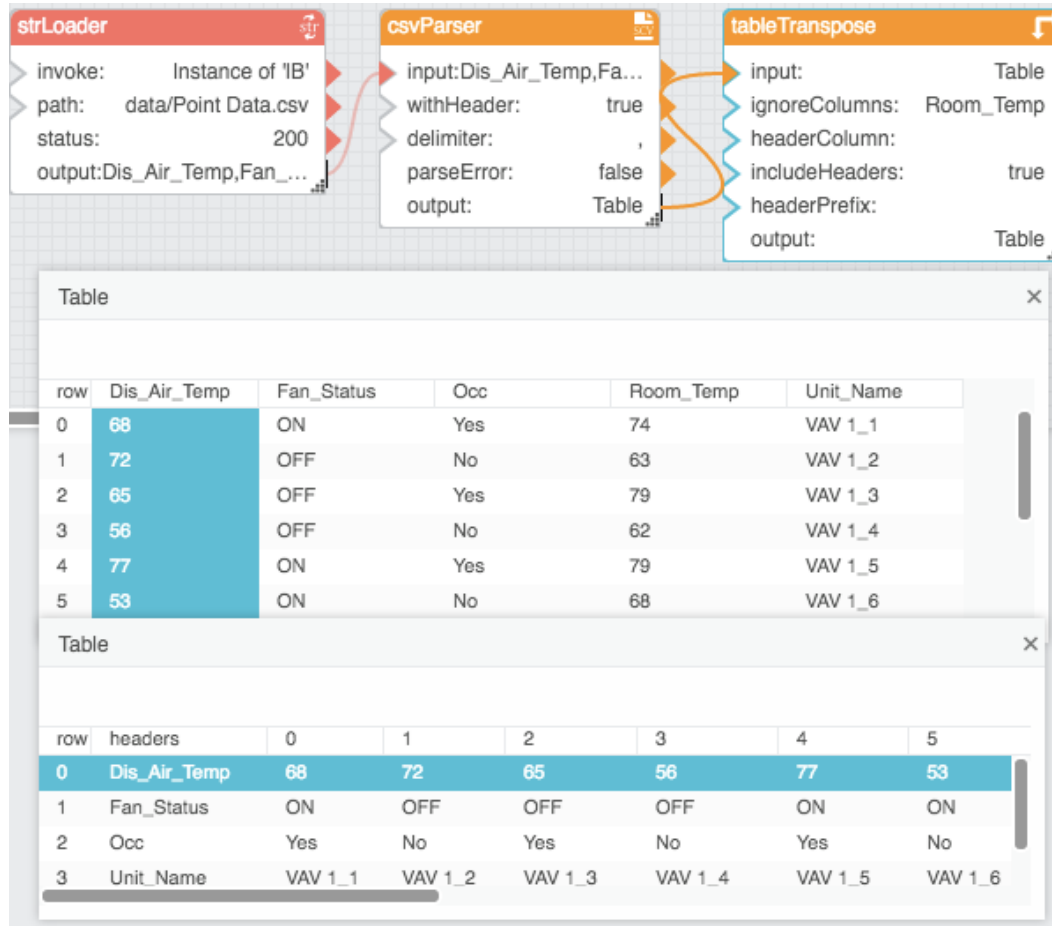
Example

Figure 170 shows an example of the Transpose block.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Figure 170. Transpose Block



Realtime Recorder

The Realtime Recorder block monitors changes to specified values and creates a table to record current and historical values. Every time one of the monitored values changes, a row is added to the table with a timestamp and all of the values.

Realtime Recorder table contents are saved only in the current session. When the browser restarts or the dataflow link restarts, the table contents are cleared.

Input/Output Properties

The following properties of the Realtime Recorder block can take input and give output.

enabled (Boolean) Determines whether the block is currently recording and creating a table.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

bufferSize (number)	Specifies the number of rows that the output table can hold. When the buffer is full, the oldest rows are deleted from memory.
reset (trigger)	Clears the table.
name <i>n</i> (string)	Specifies a column name for the value <i>n</i> values in the output table.
value <i>n</i>	Specifies a value to monitor and output in the output table.

Output Property

The following property of the Realtime Recorder block can give output but cannot take input.

output (table)	Returns a table of timestamps and values. The table contains a row for each monitored change.
-----------------------	---

Example

Figure 171 shows an example of the Realtime Recorder block. In this example, a Realtime Recorder block is creating a table that monitors any changes to an input number and to a stopwatch value.

Figure 171. Realtime Recorder Block

The screenshot shows the Dataflow Editor interface. On the left, there are three input blocks: a 'number' block with a value of 70, a 'stopwatch' block with settings (enabled: false, interval: 1, step: 1, reset: Instance of 'IB', output: 10), and a 'realtimeRecorder' block. The 'realtimeRecorder' block is configured with: bufferSize: 1024, ignoreNullValue: (unchecked), reset: Instance of 'IB', name 0: temperature, value 0: 70, name 1: stopwatch, value 1: 10, and output: Table. On the right, a 'Table' window displays the output data:

row	timestamp	temperature	stopwatch
0	2016-04-13T16:09:50.766	70	1
1	2016-04-13T16:09:51.764	70	2
2	2016-04-13T16:09:52.757	70	3
3	2016-04-13T16:09:53.766	70	4
4	2016-04-13T16:09:54.765	70	5
5	2016-04-13T16:09:55.758	70	6
6	2016-04-13T16:09:56.765	70	7
7	2016-04-13T16:09:57.760	70	8
8	2016-04-13T16:09:58.766	70	9
9	2016-04-13T16:09:59.760	70	10

Series Realtime Recorder

The Series Realtime Recorder block monitors the single current value of each specified metric and creates a table of those current values without storing historical values.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Input/Output Properties

The following properties of the Series Realtime Recorder block can take input and give output.

enabled (Boolean)	Determines whether the block is currently recording and updating the table.
name <i>n</i> (string)	Specifies a name corresponding to value <i>n</i> , to store in the output table.
value <i>n</i>	Specifies a value corresponding to name <i>n</i> , to store in the output table.

Output Property

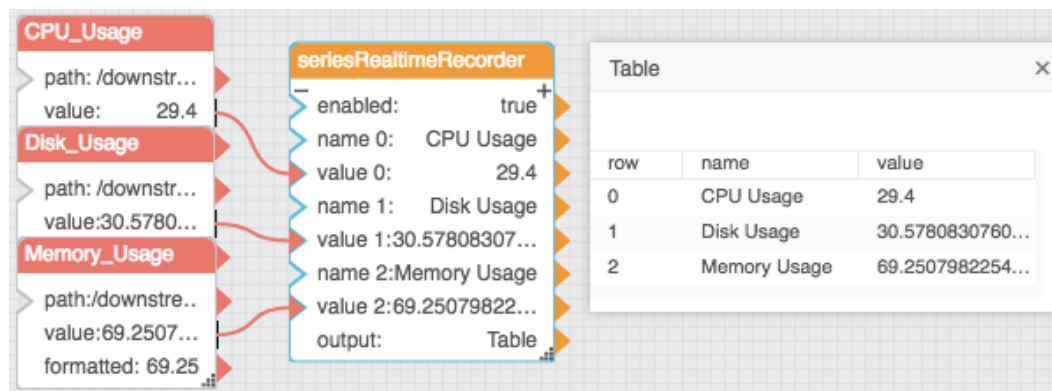
The following property of the Series Realtime Recorder block can give output but cannot take input.

output (table)	Returns a table of names and values.
-----------------------	--------------------------------------

Examples

Figure 172 demonstrates an example of the Series Realtime Recorder block. In this example, values from three Load Value blocks are bound to the Series Realtime Recorder block and then a table is created to store the three values.

Figure 172. Series Realtime Recorder Block



Date Time Operations Blocks

These Date Time Operations blocks perform formatting and other operations on dates, times, and ranges.

Date Time

The Date Time block takes multiple inputs that represent a year, month, day, hour, minute, second, and millisecond and returns the sequential serial number that represents the date and time.

The Date Time block is useful when a date is supplied in a format that the Dataflow Editor does not recognize, such as YYYYMMDD. You can use the Date Time block with other blocks, such as the Substring block, to convert the dates to a serial number that the Dataflow Editor recognizes.

Dataflow blocks

Input/Output Properties

The following properties of the Date Time block can take input and give output.

year (number)	Specifies the year. Can include one to four digits.
month (number)	Specifies the month of the year, from 1 to 12 (January to December). Can be a positive or negative integer.
day (number)	Specifies the day of the month, from 1 to 31. Can be a positive or negative integer.
hour (number)	Specifies the hour of the day, from 0 to 23. Can be a positive or negative integer.
minute (number)	Specifies the minute of the hour, from 0 to 59. Can be a positive or negative integer.
second (number)	Specifies the second of the minute, from 0 to 59. Can be a positive or negative integer.
millisecond (number)	Specifies the millisecond of the second, from 0 to 999. Can be a positive or negative integer.
isUTC (Boolean)	Controls whether to use UTC. By default, this property is FALSE and the Dataflow Editor uses your computer's time zone.

Output Property

The following property of the Date Time block can give output but cannot take input.

output (number)	Returns the sequential serial number that represents a particular date and time.
------------------------	--

Special Cases

The following are special cases for the Date Time block:

- Integers greater than the maximum add to the date or time, while integers less than the minimum subtract from the date or time. For example, with **year** equal to 2016 and **month** equal to 14, the output represents February 2017.
- All null values are treated as zero (0).

Examples

Figure 173 shows a typical example of the Date Time block. In this example, a serial number is created from the input properties.

Figure 174 shows two special cases of the Date Time block. In the leftmost Date Time block, a value of 16 for the month causes the result to be the fourth month of the following year. In the rightmost Date Time block, a value of -12 for the hour causes the result to be noon of the previous day.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Dataflow blocks

Figure 173. Date Time Block Typical Case

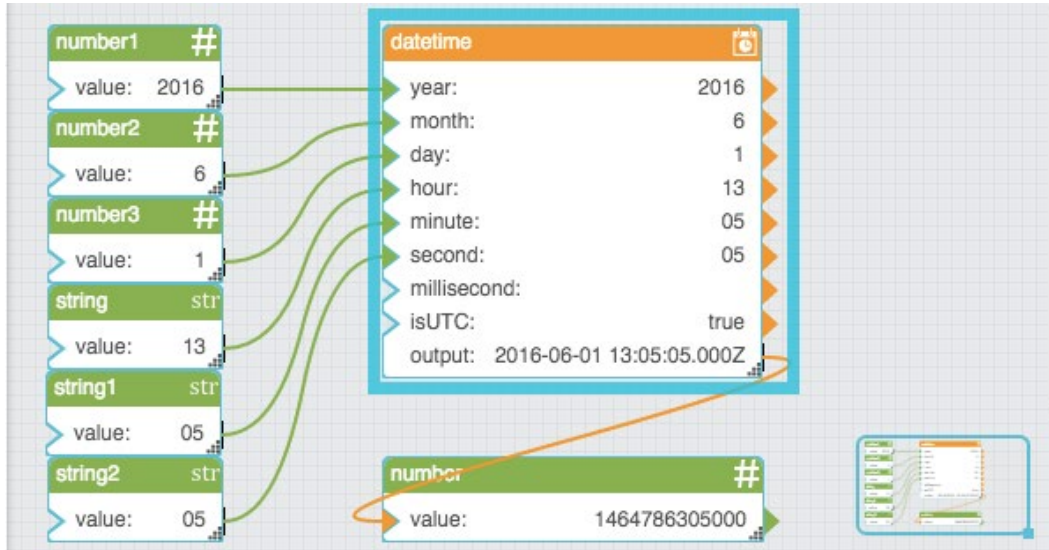


Figure 174. Date Time Block Special Cases



Date Format

The Date Format block reformats a date and time string using the defined format. This block also converts a serial number to a date and time string.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

Input/Output Properties

The following properties of the Date Format block can take input and give output.

- input** (number or string) Specifies the serial number or any supported date time string. This can also be a string from which a date can be parsed.
- format** (string) Specifies the date and time formatting string. For example, you can format the date to be displayed as “Mon, Jan 5, 2015” by using this string: E, MMM d, yyyy

Output Property

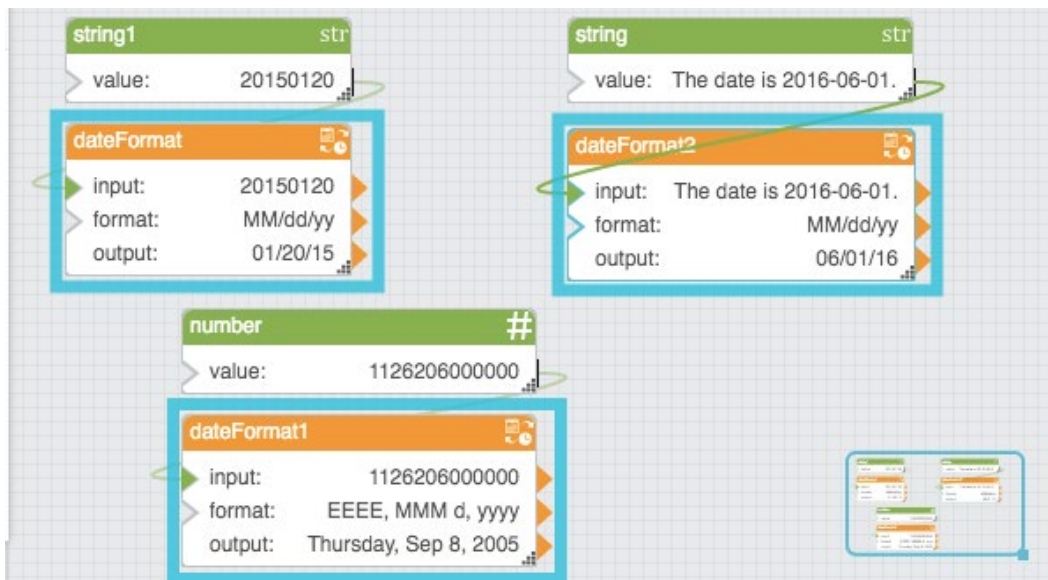
The following property of the Date Format block can give output but cannot take input.

- output** (string) Returns a date and time string created from the specified input string and format.

Examples

Figure 175 shows three examples of the Date Format block. The top left Date Format block translates a date string from one format to another format. The top right Date Format block extracts a date string from text and also reformats the date string. The bottommost Date Format block creates a date and time string from a sequential serial number.

Figure 175. Date Format Block



Parse Date Time

The Parse Date Time block converts a serial number, or a date and time string, to multiple outputs that represent the serial number, year, month, day, hour, minute, second, millisecond, and weekday.

Dataflow blocks

Input/Output Property

The following property of the Parse Date Time block can take input and give output.

input (number or string) Specifies the serial number or any supported date time string. This can also be text from which a date can be parsed.

Output Properties

The following properties of the Parse Date Time block can give output but cannot take input.

time (number) Returns the serial number.

year (number) Returns the year.

month (number) Returns the month of the year as a number from 1 to 12.

day (number) Returns the day of the month as a number from 1 to 31.

hour (number) Returns the hour of the day as a number from 0 to 23.

minute (number) Returns the minute of the hour as a number from 0 to 59.

second (number) Returns the second of the minute as a number from 0 to 59.

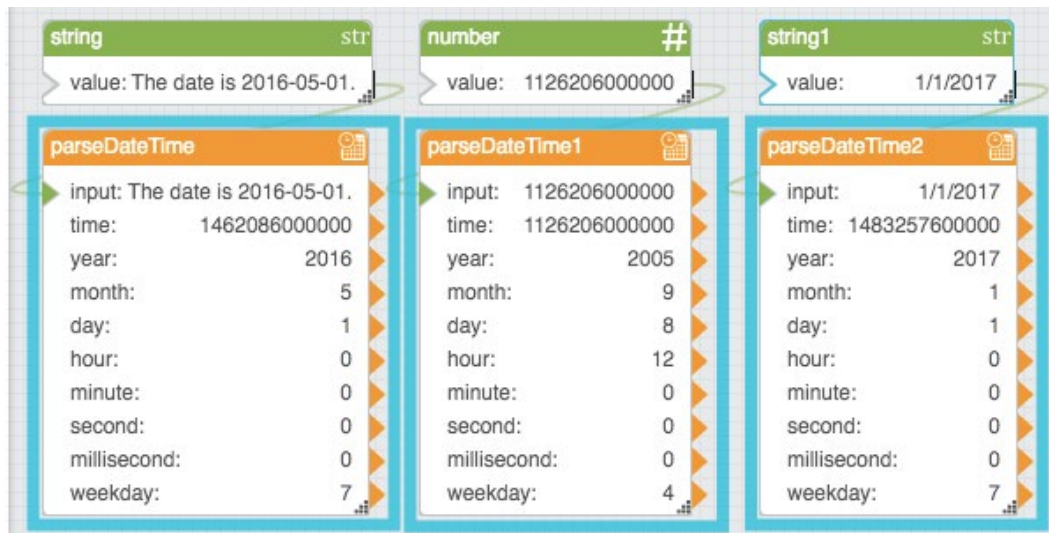
millisecond (number) Returns the millisecond of the second as a number from 0 to 999.

weekday (number) Returns the day of the week as a number from 1 to 7. The number 1 represents Sunday, and the number 7 represents Saturday.

Examples

Figure 176 shows three examples of the Parse Date Time block. The leftmost Parse Date Time block extracts a date string from text. The middle Parse Date Time block takes a serial number as input. The rightmost Parse Date Time block takes a date string as input.

Figure 176. Parse Date Time Block



Date Math

The Date Math block adds time to or subtracts time from the input date and time and returns the serial number that represents the result.

One use of the Date Math block is to calculate maturity dates. Another use is to calculate due dates that occur on the same day of the month as the date of issue.

Input/Output Properties

The following properties of the Date Math block can take input and give output.

- input** (number or date time string) Specifies the serial number or any supported date time string.
- op** (enum) Specifies the operator that the Date Math block uses to perform calculations on the input date. The value of the **op** property can be one of the following:
 - After**—The values of the other inputs are added to the input value.
 - Before**—The values of the other inputs are subtracted from the input value.
- year** (number) Specifies the number of years to add or subtract, as a number with one to four digits.
- month** (number) Specifies the number of months to add or subtract
- day** (number) Specifies the number of days to add or subtract.
- hour** (number) Specifies the number of hours to add or subtract.

Kinetic - Edge & Fog Fabric Processing Module

Dataflow Editor User Guide

Dataflow blocks

minute (number)	Specifies the number of minutes to add or subtract.
second (number)	Specifies the number of seconds to add or subtract.
millisecond (number)	Specifies the number of milliseconds to add or subtract.

Output Property

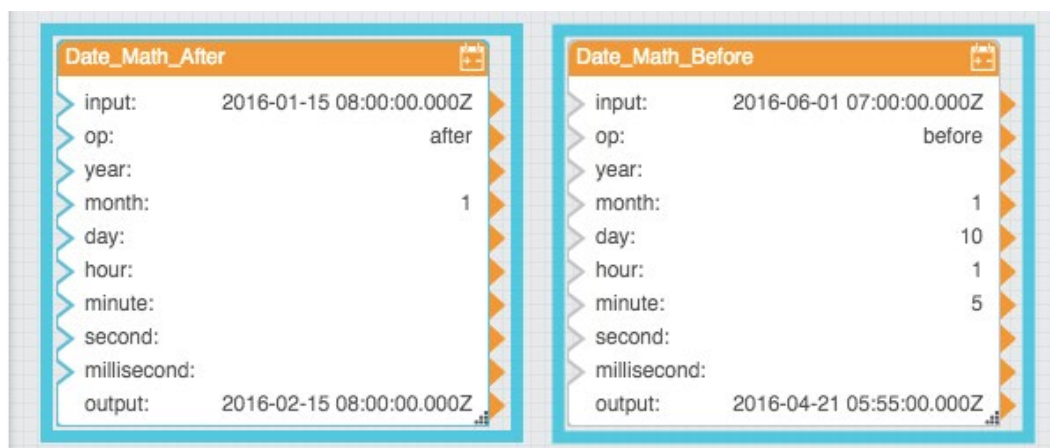
The following property of the Date Math block can give output but cannot take input.

output (number)	Returns the result of the operation.
------------------------	--------------------------------------

Examples

Figure 177 shows two examples of the Date Math block. In the leftmost Date Math block, one month is added to the input date. In the rightmost Date Math block, one month, ten days, one hour, and five minutes are subtracted from the input date.

Figure 177. Date Math Block



Date Range

The Date Range block returns a formatted date and time range based on the defined formatting options.

Special Values

Special values can be the following strings:

- today
- yesterday
- thisWeek
- lastWeek

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Dataflow blocks

- thisMonth
- lastMonth
- thisYear
- lastYear

Input/Output Properties

The following properties of the Date Range block can take input and give output.

input (date range)	Specifies the currently selected range. You can select a range by clicking the property and using the popup window.
next (trigger)	Advances the value of the input property to the next range of the same duration.
previous (trigger)	Changes the value of the input property to the previous range of the same duration.
format (string)	Specifies the date and time formatting string that is used to format the dates and times in this block.
rangeDelimiter (string)	Specifies the string used to delimit the beginning of the range and the end of the range.
keepSpecial (enum)	Specifies when a special value is used as the output properties versus when the values of the format and delimiter properties are used. The value of the keepSpecial property can be one of the following: None —The format and delimiter properties are always used, if defined. Raw —The special value is used, if applicable. Otherwise, the format and delimiter properties are used. Formatted —The special value is used, if applicable. Short default formats are automatically used when available. The format and delimiter properties are used when necessary. Format Clean —The special value is used, if applicable. Shortened versions of the format property are used when available.

Output Properties

The following properties of the Date Range block can give output but cannot take input.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Constants Blocks

formatted (string)	Returns the range, formatted according to the format , rangeDelimiter , and keepSpecial properties.
nextRange (date range)	Returns the next range of the same duration after input .
thisRange (date range)	Returns the currently selected range in input .
previousRange (date range)	Returns the previous range of the same duration before input .
duration (number)	Returns the duration of the currently selected range, in milliseconds.

Example

Figure 178 demonstrates the Block Properties pane for an example of the Date Range block. In this example, a one-week range has been selected and a format has been defined.

Figure 178. Date Range Block

dateRange	
input	thisWeek
formatted	5/1/2016 to 5/7/2016
nextRange	2016-05-08T00:00:00/2016-05-14T23:59:59
thisRange	2016-05-01T00:00:00/2016-05-07T23:59:59
previousRange	lastWeek
next	Invoke
previous	Invoke
format	yMd
rangeDelimiter	to
keepSpecial	Formatted
duration	604800000

Constants Blocks

These Constants blocks are special Number blocks that hold mathematical constants when the blocks are created.

Pi

The Pi block is a Number block. When the Pi block is created, the Pi block holds the mathematical constant pi.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Constants Blocks

Input/Output Property

The following property of the Pi block can take input and give output.

value (number) Sets and returns the number held by this Pi block. When a Pi block is created, the **value** property holds the mathematical constant pi.

Example

Figure 179 demonstrates an example of the Pi block.

Figure 179. Pi Block



E

The E block is a Number block. When the E block is created, the E block holds the mathematical constant e.

Input/Output Property

The following property of the E block can take input and give output.

value (number) Sets and returns the number held by this E block. When an E block is created, the **value** property holds the mathematical constant e.

Example

Figure 180 demonstrates an example of the E block.

Figure 180. E Block



SQRT2

The SQRT2 is a Number block. When the SQRT2 block is created, the SQRT2 block holds the mathematical constant for the square root of two.

Input/Output Property

The following property of the SQRT2 block can take input and give output.

value (number) Sets and returns the number held by this SQRT2 block. When a SQRT2 block is created, the **value** property holds the mathematical constant for the square root of two.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Constants Blocks

Example

Figure 181 demonstrates an example of the SQRT2 block.

Figure 181. SQRT2 Block



LN2

The LN2 block is a Number block. When the LN2 block is created, the LN2 block holds the mathematical constant for the natural logarithm of two.

Input/Output Property

The following property of the LN2 block can take input and give output.

value (number)	Sets and returns the number held by this LN2 block. When the LN2 block is created, the value property holds the mathematical constant for the natural logarithm of two.
-----------------------	--

Example

Figure 182 demonstrates an example of the LN2 block.

Figure 182. LN2 Block



LN10

The LN10 block is a Number block. When the LN10 block is created, the LN10 block holds the mathematical constant for the natural logarithm of ten.

Input/Output Property

The following property of the LN10 block can take input and give output.

value (number)	Sets and returns the number held by this LN10 block. When the LN10 block is created, the value property holds the mathematical constant for the natural logarithm of ten.
-----------------------	--

Example

Figure 183 demonstrates an example of the LN10 block.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Constants Blocks

Figure 183. LN10 Block



LOG2E

The LOG2E block is a Number block. When the LOG2E block is created, the LOG2E block holds the mathematical constant for the base-2 logarithm of the constant e .

Input/Output Property

The following property of the LOG2E block can take input and give output.

value (number)	Sets and returns the number held by this LOG2E block. When a LOG2E block is created, the value property holds the mathematical constant for the base-2 logarithm of the constant e .
-----------------------	---

Example

Figure 184 demonstrates an example of the LOG2E block.

Figure 184. LOG2E Block



LOG10E

The LOG10E block is a Number block. When the LOG10E block is created, the LOG10E block holds the mathematical constant for the base-10 logarithm of the constant e .

Input/Output Property

The following property of the LOG10E block can take input and give output.

value (number)	Sets and returns the number held by this LOG10E block. When a LOG10E block is created, the value property holds the mathematical constant for the base-10 logarithm of the constant e .
-----------------------	--

Example

Figure 185 demonstrates an example of the LOG10E block.

Figure 185. LOG10E Block



Appendixes

These appendixes are useful when you write script in the Dataflow Editor.

Appendix 1: Dataflow Editor Script

You can use script in the Script, JavaScript, Filter, and Column Mapping dataflow blocks.

You can also use the Dataflow Editor syntax in bindings.

Although Dataflow Editor script has some differences from JavaScript, JavaScript can be used to accomplish many custom calculation tasks in the dataflow.

The differences between JavaScript and Dataflow Editor script are as follows:

- In Dataflow Editor script, `list` does not support dynamic properties.
- The Dataflow Editor uses the `DateTime` type instead of the JavaScript `Date` type.

Appendix 2: Dataflow Editor Syntax

When you use script in a Script, Filter, or Column Mapping block, it is important to understand Dataflow Editor syntax. The script execution context and the block location both have an important relationship with the model that this syntax describes.

Use the at sign (@) to interact with elements in the dataflow model. For example:

- `@` refers to the current block
- `@parent` refer to the logical parent of the current block. You can use this string more than once to refer to further ancestors, such as `@parent.@parent.@parent`.
- `@params` refers to the parameters of the symbol being referred to. For example, when used inside a dataflow symbol, in the symbol's root dataflow model, `@parent.@params` refers to the parameters of this symbol.
- In a binding, `@parent.@table.0_v1` refers to the cell in row zero, column v1, for a table that shares the same parent as the current block.
- `@<name>` refers to the Dataflow Editor syntax name of the block at the specified path. For example, to refer to the value property of a String block, you might use `@parent.@string1.value`.

Viewing block types and Dataflow Editor syntax names

To view the Dataflow Editor syntax name of a property and that property's parent:

1. Hover over the property until a blue dot appears.
2. Hold the Alt or Option key while right-clicking the blue dot.

A tooltip displays all of the following:

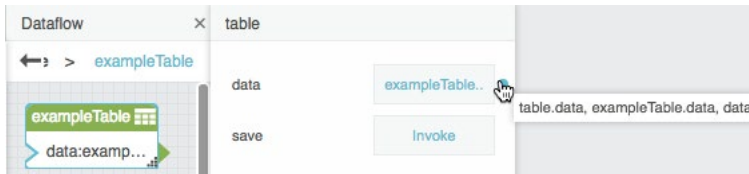
Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Appendixes

- Parent element type and property name
- Parent element name in Dataflow Editor syntax and property name
- Property name

The following images demonstrate this tooltip.

Figure 186. Viewing Block Type and Name



Accessing and setting values using script

The following example demonstrates how to use script to access and set a value.

```
//Sets the "a" property of the current block to
//the value of the string block located on
//the parent dataflow.
@.a = @parent.string.value;
```

Use = to set a runtime value for an object property and use ~= to set a value that will be serialized as part of a subsequent save. The following examples demonstrate how to set both kinds of values.

```
//Sets the runtime "value" property of the
//parent object's "string" block to
//"stringValue".
@parent.string.value = "stringValue";

//Sets the serialized "value" property of the
//parent object's "string" block to
//"stringValue".
@parent.string.value ~= "stringValue";
```

Creating bindings using script

To create a binding using script, use <targetPropertyPath> ~= ['<sourcePropertyPath>']. The following examples demonstrate how to create bindings using script.

```
//Creates a serialized binding of the value of
//table1 to the value of table2.
@parent.table2.value ~= '[@parent.table1.value]'
```

Example

Figure 187 shows a dataflow model before invoking a script. This script sets values, creates a binding, and returns an output value.

Kinetic - Edge & Fog Fabric Processing Module
Dataflow Editor User Guide

Appendixes

Figure 187. Dataflow Model before invoking script

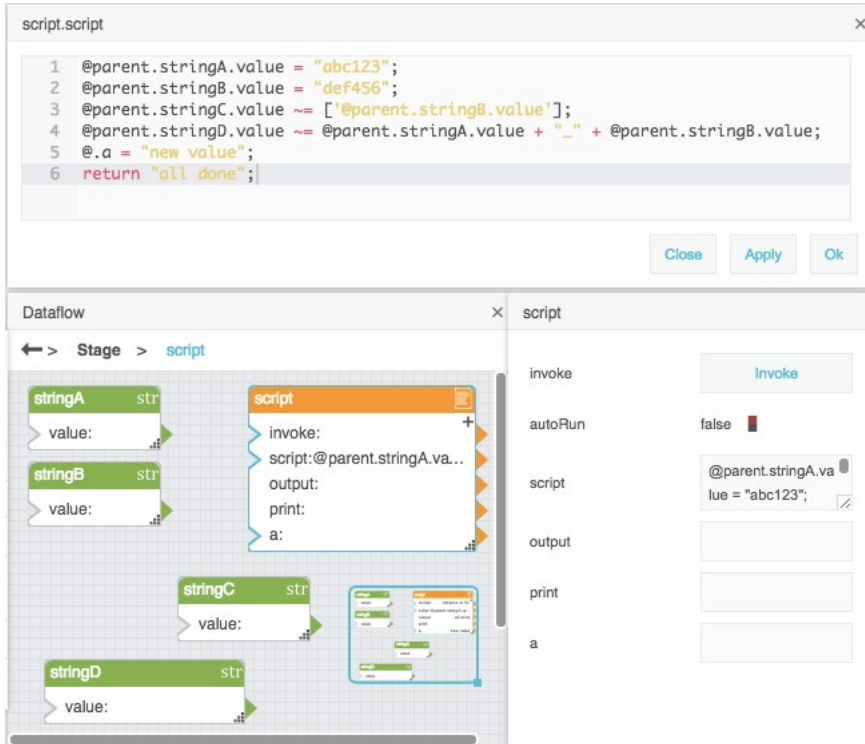
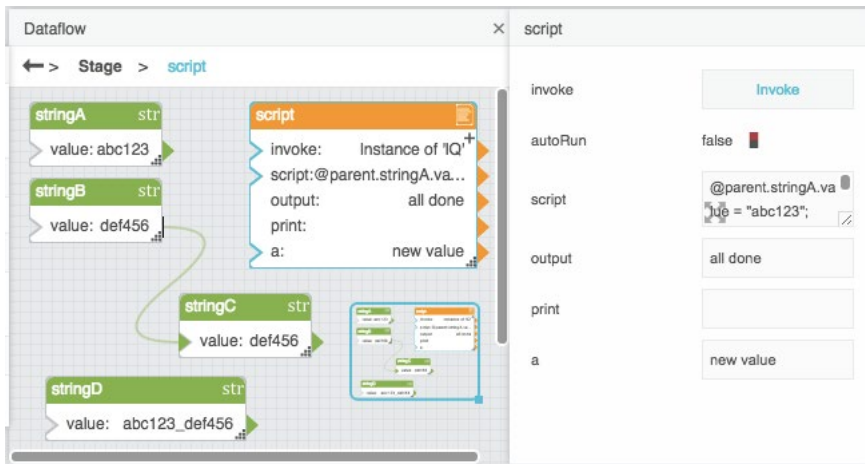


Figure 188 shows the same dataflow model after invoking the script.

Figure 188. Dataflow Model after invoking script



Appendix 3: Scripts for debugging

The following example prints the results of an expression to the current block's **print** property:

```
//Prints 2 + 2 = 4  
print("2 + 2 = ", 2 + 2);
```

The following example prints the results of expressions to the browser console:

```
/*Prints to browser's console:  
1  
2  
3  
4  
5*/  
json = JSON.parse("[1,2,3,4,5]");  
for (var i = 0; i < json.length; i++) {  
    printConsole(json[i]); }  
*/
```

Appendix 4: Reusing script

You can refer to a script from within another Script, Column Mapping, or Filter block. When you do this, you can use `$params`, which is a special list that is only available when a script is called as a function from another script.

The following example creates a script and then calls it from a Script block and a Column Mapping block:

1. Create a Script block and change its Dataflow Editor syntax name to `multiplyAndAdd`.
2. For the script property of `multiplyAndAdd`, add a script that multiplies two `$params` values and adds some other variable to the product:

```
return $params[0] * params[1] + @.a;
```

For this example, you must also add an **a** property to `multiplyAndAdd` by clicking the plus sign and then set the value of **a** as 1.

3. Add a second Script block, and change its name to `callFunction`.

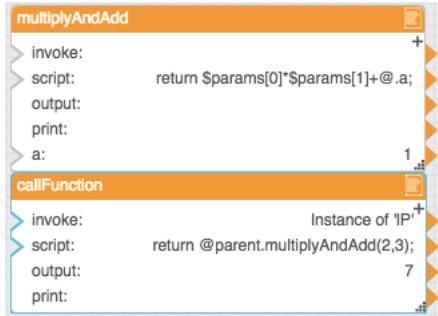
For the script property of `callFunction`, add a script that calls `multiplyAndAdd` and provides some parameter values:

```
return@parent.multiplyAndAdd\(2,3\);
```

In this example, `callFunction` returns 7, as shown in the following image.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

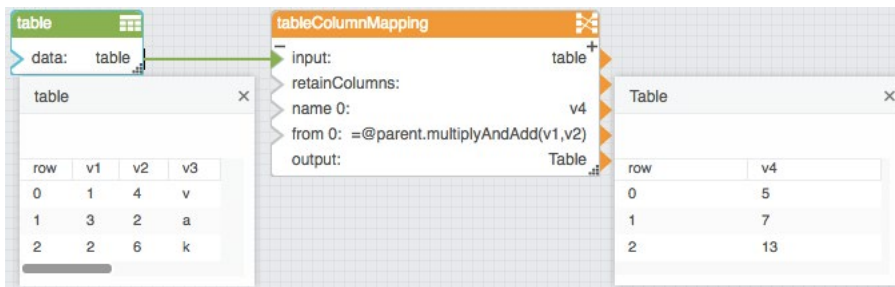
Appendixes



4. Add a Table block and Column Mapping block.
5. Bind the Table block value to the Column Mapping block input.
6. For **name 0**, enter `v4`.
7. For **from 0**, enter script that calls `multiplyAndAdd`:

```
=@parent.multiplyAndAdd(v1, v2)
```

In each row, the new column contains the product of **v1** and **v2** in this row, plus the value of **@.a** in `multiplyAndAdd`. The following image demonstrates the input and output tables.



Appendix 5: Operators, Formats, and Functions

This section demonstrates some operators, formats, and functions that can be used in the Dataflow Editor. For example, the Column Mapping block, Filter block, and Date Time Operations blocks use some of these items.

Operators

These operators can be used in the Dataflow Editor.

String Operators

You can use the following string operator:

- A plus sign (+) concatenates two strings.

Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

Appendixes

Number Operators

You can use the following number operators:

- A plus sign (+) adds two numbers.
- A hyphen (-) subtracts two numbers.
- An asterisk (*) multiplies two numbers.
- A forward slash (/) divides two numbers and returns the result.
- A percent symbol (%) performs a modulo operation. This operation divides two numbers and returns the remainder.

Logical Operators

Table 1 lists logical operators. Logical operators appear in expressions that can be evaluated to TRUE or FALSE.

Table 1. Logical Operators

Operator	Example
Less Than (<)	The expression <code>5<10</code> evaluates to TRUE.
Greater Than (>)	The expression <code>10>5</code> evaluates to TRUE.
Less Than or Equal To (<=)	The expression <code>5<=5</code> evaluates to TRUE.
Greater Than or Equal To (>=)	The expression <code>5>=5</code> evaluates to TRUE.
Equal To (==)	The expression <code>'5'==5</code> evaluates to TRUE.
Strictly Equal To (===)	The expression <code>5==='5'</code> evaluates to FALSE.
Not Equal To (!=)	The expression <code>'1'!=1</code> evaluates to FALSE.
Not Strictly Equal To (!==)	The expression <code>'1'!==1</code> evaluates to TRUE.
And (&&)	The expression <code>true && false</code> evaluates to FALSE.
Or ()	The expression <code>true false</code> evaluates to TRUE.

Bitwise Operators

You can use the following bitwise operators:

- Shift Left (<<)
- Shift Right (>>)
- Bitwise And (&)
- Bitwise Or (|)

Appendixes

- Bitwise Xor (^)

String Usage and Functions

String Literals

The following are string literals:

- 'Single quotes'
- " Double quotes"
- 'Double quotes in " single" quotes'
- " Single quotes in 'double' quotes"
- " escape quote \" "

Reserved Characters

In strings, you can use a backslash (\) to escape the following reserved characters:

- Single quotation mark (')
- Double quotation mark (")
- Backslash (\)

Special sequences

In strings, you can use the following special character sequences to create these effects:

- \n creates a new line.
- \r creates a carriage return.
- \t creates a tab.
- \u<code point> uses a Unicode code point.

String Functions

The following functions perform an operation on an input string.

String(obj) Takes an input object, casts the object as a string, and returns the string.

For example, the following function returns **123** as a string, not a number.

```
String(123);
```

Appendixes

String(num,16) Takes an input base-10 number, converts the number to base 16, and returns the base-16 number.

For example, the following function returns the base-16 number 1b:

```
String(27,16);
```

str.length Counts the number of characters in an input string and returns the number.

For example, the following function returns the number 4:

```
"acbd".length;
```

str.charAt(pos) Returns the character at the specified position in an input string. The first position is represented as 0.

For example, the following function returns the character b:

```
"abc".charAt(1);
```

str.charCodeAt(pos) Returns the character code for the character at the specified position in an input string. The first position is represented as 0.

For example, the following function returns the code 98, the character code for the letter "b":

```
"abc".charCodeAt(1);
```

str.indexOf(searchStr) Returns the position of the first character of the first occurrence of the search string. Returns -1 if the search string is not found. The first position is represented as 0.

For example, the following function returns the number 1.

```
"abc".indexOf("bc");
```

str.lastIndexOf(searchStr) Returns the position of the first character of the last occurrence of the search string. Returns -1 if the search string is not found. The first position is represented with 0.

For example, the following function returns the number 2.

```
"abb".lastIndexOf("b");
```

str.split(delimiter) Takes an input string that contains delimiters and returns an array of strings.

For example, the following function returns the array ["1", "2", "3"].

```
"1,2,3".split(",");
```

Appendixes

- str = list.join(char)** Takes an input list and converts it to an output string by joining items in the list with a provided delimiting character.
- For example, the following function returns the string **1/2/3**:
- ```
[1,2,3].join("/");
```
- str.substr(start, length)** Extracts the specified number of characters starting from the specified position and returns the substring. The first position is represented as 0.
- For example, the following function returns the string **bc**:
- ```
"abcd".substr(1,2);
```
- str.substring(start,end)** Extracts the substring beginning with the specified start position and ending one position before the specified end value. The first position in a string is represented as 0.
- For example, the following function returns the string **b**:
- ```
"abcd".substring(1,2);
```
- str.replace(regexPattern, replace)** Replaces all instances of the specified regular expression pattern with the replacement string.
- For example, the following function returns the string **d d**:
- ```
"DG DG".replace(/D(\w+)/g, "d");
```
- str.replaceAll(find, replace)** Replaces all instances of the specified **find** string with the specified **replace** string.
- For example, the following function returns the string **1,2,3**:
- ```
"1#2#3".replaceAll("#", ",");
```
- str.toLowerCase()** Converts all letters in a string to lowercase letters.
- For example, the following function returns the string **dataflow**:
- ```
"DATAFLOW".toLowerCase();
```
- str.toUpperCase()** Converts all letters in a string to uppercase letters.
- For example, the following function returns the string **DATAFLOW**:
- ```
"dataflow".toUpperCase();
```

Appendixes

|                                  |                                                                                                                                                                                                                      |
|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>str.encodeUriComponent()</b>  | Encodes an input URI component.<br><br>For example, the following function returns the string <b>a%20b</b> :<br><br><pre>"a b".encodeUriComponent();</pre>                                                           |
| <b>str.decodeUriComponent();</b> | Decodes an encoded URI component string.<br><br>For example, the following function returns the string <b>a b</b> :<br><br><pre>"a%20b".decodeUriComponent();</pre>                                                  |
| <b>str.splitQuery();</b>         | Splits the input query string and decodes the string into an Object.<br><br>For example, the following function returns the object <b>{"a b": "1", "c": "d"}</b> :<br><br><pre>"a%20b=1&amp;c=d".splitQuery();</pre> |
| <b>str.encodeBase64()</b>        | Encodes the input string, using base-64 encoding.                                                                                                                                                                    |
| <b>str.decodeBase64()</b>        | Decodes a string that was encoded using <b>str.encodeBase64()</b> .                                                                                                                                                  |
| <b>str.md5()</b>                 | Generates a 16-byte hash value that identifies the input string.                                                                                                                                                     |
| <b>str.sha1()</b>                | Generates a 20-byte hash value that identifies the input string.                                                                                                                                                     |
| <b>str.sha256()</b>              | Generates an almost-unique 256-bit hash value that identifies the input string.                                                                                                                                      |

## Number Formats and Functions

### Number Format Patterns

You can specify a number format using the following subset of ICU formatting patterns:

- Zero (0) represents a single digit and is included even if the digit is a non-significant zero.
- A number sign (#) represents a single digit and is omitted if the digit is a non-significant zero.
- A decimal (.) represents the decimal separator.
- A hyphen (-) represents a minus sign.
- A comma (,) represents the grouping separator.
- The letter E separates a number and that number's exponent.
- A plus sign (+) before an exponent indicates to prefix a positive exponent with a plus sign.
- A percent symbol (%) in a prefix or suffix indicates to multiply the number by 100 and show the number as a percentage.



Appendixes

- A per mile sign (‰ or \u2030) indicates to multiply the number by 1000 and show the number as per mile.
- A currency sign (¤ or \u00A4) is replaced by the currency name.
- Single quotes (') are used to quote special characters.
- A semicolon (;) separates the positive and negative patterns if both are present.

**Number Functions**

**Number(str)**

Takes an input string, converts the string to a number, and returns the number.

For example, the following function returns 12.12 as a number, not a string:

```
Number("12.12");
```

**isNaN(str)**

Returns TRUE if the string is not a number.

For example, the following function returns TRUE:

```
isNaN(Number("abc"));
```

**num = parseNumber(str)**

Parses a number from a string and returns the number.

For example, the following function returns the number 12.11:

```
parseNumber('$12.11');
```

**str = numberFormat(num, pattern)**

Formats the input number using the input number format pattern.

For example, the following function returns the string \$123.46:

```
numberFormat(123.456789,"$#.00");
```

**Math Functions**

You can use the following functions to perform mathematical operations:

- **Math.abs(num)**—Returns the absolute value of **num**.
- **Math.min(num1,num2,...)**—Returns the smallest input number.
- **Math.max(num1,num2,...)**—Returns the greatest input number.
- **Math.sin(num)**—Returns the sine of **num**.
- **Math.cos(num)**—Returns the cosine of **num**.
- **Math.tan(num)**—Returns the tangent of **num**.
- **Math.asin(num)**—Returns the arcsine of **num**.

## Appendixes

- **Math.acos(num)**—Returns the arccosine of **num**.
- **Math.atan(num)**—Returns the arctangent of **num**.
- **Math.atan2(y, x)**—Returns the arctangent of the quotient of **y** divided by **x**.
- **Math.ceil(num)**—Rounds up **num** to the nearest integer.
- **Math.floor(num)**—Rounds down **num** to the nearest integer.
- **Math.round(num)**—Rounds **num** up or down to the nearest integer.
- **Math.exp(num)**— Returns  $e^x$  where  $x$  is **num**.
- **Math.log(num)**—Returns the base- $e$  logarithm of **num**.
- **Math.sqrt(num)**—Returns the square root of **num**.
- **Math.pow(num, exponent)**—Returns  $x^y$  where  $x$  is **num** and  $y$  is **exponent**.
- **Math.random()**—Returns a random number between zero and one.

## Constants

The following functions return constant values:

- **Math.PI**—Returns pi.
- **Math.E**—Returns  $e$ .
- **Math.LN2**—Returns the natural logarithm of 2.
- **Math.LN10**—Returns the natural logarithm of 10.
- **Math.LOG2E**—Returns the base-2 logarithm of  $e$ .
- **Math.LOG10E**—Returns the base-10 logarithm of  $e$ .
- **Math.SQRT2**—Returns the square root of 2.
- **Math.SQRT1\_2**—Returns the square root of one half.

## Date and Time Formats and Functions

The following are examples of date and time strings that are formatted using date and time formatting strings:

- 2017-02-27 13:27:00
- 2017-02-27 13:27:00.123456z
- 20170227 13:27:00
- 20170227T132700

## Kinetic - Edge & Fog Fabric Processing Module Dataflow Editor User Guide

### Appendixes

- 20170227
- +20170227
- 2017-02-27T14Z
- 2017-02-27T14+00:00

**Note:** The string 2017-02-27T14:00:00-0500 represents the same date and time as the string 2017-02-27T19:00:00Z.

### Supported Date and Time Formatting String Components

Table 2 lists date and time pattern components. These components can be included within a date and time format string.

#### Notes:

- Date and time strings are affected by locale.
- In some languages, the month of a year or day of a week is a different word or declension from a standalone month or day.

**Table 2. Supported Date and Time Formatting String Components**

| Formatting String Component | Meaning                                | Example |
|-----------------------------|----------------------------------------|---------|
| G                           | Era designator                         | AD      |
| y                           | Year                                   | 2017    |
| M                           | Month in year, number                  | 8       |
| MM                          | Month in year, number as two digits    | 08      |
| MMM                         | Month in year, abbreviated name        | Aug     |
| MMMM                        | Month in year, full name               | August  |
| L                           | Standalone month, number               | 8       |
| LL                          | Standalone month, number as two digits | 08      |
| LLL                         | Standalone month, abbreviated name     | Aug     |
| LLLL                        | Standalone month, full name            | August  |
| d                           | Day in month                           | 10      |
| c                           | Standalone day                         | 10      |
| h                           | Hour in 12-hour time (1-12)            | 12      |

**Kinetic - Edge & Fog Fabric Processing Module**  
**Dataflow Editor User Guide**


---

 Appendixes

|                           |                                          |                       |
|---------------------------|------------------------------------------|-----------------------|
| H                         | Hour in 24-hour time (0-23)              | 13                    |
| m                         | Minute in hour                           | 30                    |
| s                         | Second in minute                         | 55                    |
| S                         | Fractional second                        | 978                   |
| E                         | Day of week, abbreviated name            | Fri                   |
| EEEE                      | Day of week, full name                   | Friday                |
| D                         | Day of year                              | 189                   |
| a                         | AM or PM designator                      | PM                    |
| k                         | Hour of day, in 24-hour time (0-23)      | 13                    |
| K                         | Hour of day, in 12-hour time (0-11)      | 0                     |
| z                         | General time zone                        | Pacific Standard Time |
| Z                         | Time zone, RFC 822                       | -0800                 |
| v                         | Time zone, generic name                  | Pacific Time          |
| Q                         | Quarter                                  | Q3                    |
| single quotation mark (') | Escapes a character or encloses a string | 'o'clock'             |

**Supported ICU dateFormat Skeleton Components**

Table 3 lists ICU date and time skeleton components. These components can be combined to create an ICU dateFormat skeleton.

**Notes:**

- Date and time strings are affected by locale.
- You can use two vertical bar characters separated by a space (| |) to join two dateFormat skeletons. For example, the string **yMd|Hms** produces the string **8/24/2015 15:38:00**.

**Table 3. Supported ICU dateFormat Skeleton Components**

| ICU Skeleton Component | Meaning      | Example (en_us locale) |
|------------------------|--------------|------------------------|
| d                      | Day of month | 24                     |
| E                      | Day of week  | Thu                    |

**Kinetic - Edge & Fog Fabric Processing Module**  
**Dataflow Editor User Guide**


---

 Appendixes

| ICU Skeleton Component | Meaning                           | Example (en_us locale)    |
|------------------------|-----------------------------------|---------------------------|
| EEEE                   | Day of week                       | Thursday                  |
| LLL                    | Standalone month                  | Aug                       |
| LLLL                   | Standalone month                  | August                    |
| M                      | Month                             | 8                         |
| Md                     | Month and day                     | 8/24                      |
| Med                    | Month, day, and day of week       | Thu, 8/24                 |
| MMM                    | Month                             | Aug                       |
| MMMd                   | Month and day                     | Aug 24                    |
| MMMEd                  | Month, day, and day of week       | Thu, Aug 24               |
| MMMM                   | Month                             | August                    |
| MMMMd                  | Month and day                     | August 24                 |
| MMMMEEEEd              | Month, day, and day of week       | Thursday, August 24       |
| QQQ                    | Quarter                           | Q3                        |
| QQQQ                   | Quarter                           | 3rd quarter               |
| y                      | Year                              | 2015                      |
| yM                     | Year and month                    | 8/2015                    |
| yMd                    | Year, month, and day              | 8/1/2015                  |
| yMEd                   | Year, month, day, and day of week | Thu, 8/24/2015            |
| yMMM                   | Year and month                    | Aug 2015                  |
| yMMMd                  | Year, month, and day              | Aug 1, 2015               |
| yMMMEd                 | Year, month, day, and day of week | Thu, Aug 24, 2015         |
| yMMMM                  | Year and month                    | August 2015               |
| yMMMMd                 | Year, month, and day              | August 24, 2015           |
| yMMMMEEEEd             | Year, month, day, and day of week | Thursday, August 24, 2015 |

| ICU Skeleton Component | Meaning                                 | Example (en_us locale) |
|------------------------|-----------------------------------------|------------------------|
| yQQQ                   | Year and quarter                        | Q3 2015                |
| yQQQQ                  | Year and quarter                        | 3rd quarter 2015       |
| H                      | Hour in day (24-hour time)              | 15                     |
| Hm                     | Hour and minute (24-hour time)          | 15:38                  |
| Hms                    | Hour, minute, and second (24-hour time) | 15:38:00               |
| j                      | Hour (12-hour time)                     | 3 PM                   |
| jm                     | Hour and minute (12-hour time)          | 3:38 PM                |
| jms                    | Hour, minute, and second (12-hour time) | 3:38:00 PM             |
| m                      | Minute                                  | 38                     |
| ms                     | Minute and second                       | 38:00                  |
| s                      | Second                                  | 0                      |

## Date and Time Functions

The following functions perform operations relating to date and time.

**new DateTime()** Returns a DateTime object that is initialized with the current date and time.

For example, the following function returns the current date and time:

```
(new DateTime());
```

**new DateTime (millisecondsSinceEpoch)** Returns a DateTime object that is initialized with the specified value.

For example, the following example returns **1969-12-31 16:00:00.000**:

```
new DateTime(0);
```

**new DateTime(dateString)** Returns a DateTime object that is initialized with the parsed date and time string.

For example, the following function returns **2017-02-27 13:27:00.000**:

```
new DateTime('2017-02-27 13:27:00');
```

### Appendixes

**new  
DateTime(y,m,d,h,m,s,ms)** Returns a DateTime object that is initialized with the specified values. For example, the following function returns **2017-01-01 00:00:00.000**:

```
new DateTime(2017,1,1,0,0,0,0);
```

**day** Returns a day of the month.

For example, the following function returns 1.

```
new DateTime(2017,1,1).day;
```

**month** Returns a month of the year. January is represented as 1.

The following function returns 2.

```
new DateTime('2017-02-27 13:27:00').month;
```

**year** Returns a year.

The following function returns the current year:

```
new DateTime().year;
```

**hour** Returns an hour.

The following function returns 13.

```
new DateTime('2017-02-27 13:27:00').hour;
```

**minute** Returns a minute.

The following function returns 27.

```
new DateTime('2017-02-27 13:27:00').minute;
```

**second** Returns a second.

The following function returns 0.

```
new DateTime('2017-02-27 13:27:00').second;
```

**millisecond** Returns a millisecond.

The following function returns 798.

```
new DateTime('2017-02-27T13:27:00.798').millisecond;
```

**millisecondsSinceEpoch** Returns the milliseconds since the start of the Unix epoch.  
The following function returns the `millisecondsSinceEpoch`, such as 1466113382710.

```
new DateTime().millisecondSinceEpoch;
```

### Appendixes

|                                     |                                                                                                                                                                                                                                                                                                           |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>weekday</b>                      | Returns a day of the week. Monday is represented as 1 and Sunday is represented as 7. For example, the following function returns 7:<br><br><pre>new DateTime(2016,5,8).weekday;</pre>                                                                                                                    |
| <b>isUtc</b>                        | Returns whether the date is UTC.                                                                                                                                                                                                                                                                          |
| <b>dateNow()</b>                    | Returns the <code>millisecondsSinceEpoch</code> of the current date and time.                                                                                                                                                                                                                             |
| <b>dateParse(str)</b>               | Returns <code>int(millisecondsSinceEpoch)</code> of a parsed string. For example, the following function returns <b>1293868800000</b> :<br><br><pre>dateParse('1/1/2011');</pre> <p><b>Note:</b> <code>dateParse</code> supports a superset of formats supported by <code>DateTime(dateString)</code></p> |
| <b>dateFormat(date, dateFormat)</b> | Parses date string or uses <code>millisecondsSinceEpoch</code> and returns the string in the specified <code>dateFormat</code> . For example, the following function returns the current month and year as a string, such as 08/2016.<br><br><pre>dateFormat(dateNow(), "MM/yyyy");</pre>                 |

### List Functions

The following functions perform operations on an input string.

|               |                                                                                                                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>length</b> | Returns the number of items in a list.<br><br>For example, the following function returns the number 3:<br><br><pre>[1,2,3].length;</pre>                                                                                              |
| <b>push()</b> | Adds an item to the end of a list.<br><br>For example, the following example returns the list [1,2,3]:<br><br><pre>row = [1,2]; row.push(3); return row;</pre>                                                                         |
| <b>pop</b>    | Removes an item from the end of a list and returns the item.<br><br>For example, in the following example, the value of <b>a</b> is 3 and the value of <b>b</b> is [1,2].<br><br><pre>row = [1,2,3]; @.a = row.pop(); @.b = row;</pre> |



### Appendixes

**indexOf** Returns the position of the first character of the first occurrence of the search string. Returns -1 if the search string is not found. The first position is represented as 0.

For example, the following function returns 1:

```
["a", "b", "c"].indexOf("b");
```

**join(delimiter)** Returns a delimited string of items in the list.

For example, the following function returns the string **a and b and c**:

```
["a", "b", "c"].join(" and ");
```

**splice(start, len, newItem1, newItem2,...)** Removes *n* items beginning at the **start** position, where *n* = **len**. Then, inserts new items into the same position and returns the items that are removed. The first position is represented with 0.

In the following example, the value of a is [2,3] and the value of b is [1,0,0,4]:

```
row = [1,2,3,4]; @.a = row.splice(1,2,0,0); @.b = row;
```

## JSON Functions

The following functions convert an object to JSON or convert JSON to an object.

**JSON.stringify(obj)** Converts an object to a JSON string.

The following function returns the string **{"a":1}**.

```
JSON.stringify({a:1});
```

**JSON.parse(str)** Converts a JSON string to an object. If the object is sent to an output property of the Script dataflow block, the object is converted to a string, and the block property holds a string.

The following example returns the JSON object **{"a": 1}**:

```
JSON.parse('{"a":1}');
```

## XML Functions

The following functions affect XML input.

**XML.parse(str)** Converts an XML string into XML elements. If this object is sent to an output property of the Script dataflow block, the object is converted to a string, and the block property holds a string.

The following example returns the XML object **<p id=" abc123" >def456</p>**:

```
var xml = XML.parse('<p id="a">b</p>');
```

**XML.stringify(xmlElement)** Converts XML elements into an XML string.

The following example returns the string **<p id=" a" >b</p>**:

```
var xml = XML.parse('<p id="a">b</p>');
@a = XML.stringify(xml);
```

**query(str)** Returns the first matching element.

The following example returns **<item>ITEM 1</item>**:

```
var xml = XML.parse(
 '<items><item>ITEM 1</item><item>
 ITEM 2</item></items>'
);
xml.query('item');
```

**queryAll(str)** Returns the list of matching elements.

The following example returns **<item>ITEM 1</item>**, **<item>ITEM 2</item>**:

```
var xml = XML.parse(
 '<items><item>ITEM 1</item><item>
 ITEM 2</item></items>'
);

var items = xml.queryAll('item');
@a = items.join(',');
```

**children** Returns the list of all children.

The following example returns **B 1,<c>C 1</c>**:

```
var xml = XML.parse(
 '<a>B 1<c>C 1</c>B 2'
);
var ch = xml.query('b').children;
return ch.join(',');
```

## Appendixes

|                          |                                                                                                                                                                                                                                                                                                     |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>elements</b>          | <p>Returns a list of children elements.</p> <p>The following example returns <b>&lt;c&gt;C 1&lt;/c&gt;</b>:</p> <pre>var xml = XML.parse(   '&lt;a&gt;&lt;b&gt;B 1&lt;c&gt;C 1&lt;/c&gt;&lt;/b&gt;&lt;b&gt;B 2&lt;/b&gt;&lt;/a&gt;' ); var el = xml.query('b').elements; return el.join(',');</pre> |
| <b>name</b>              | <p>Returns the name of an element.</p> <p>The following example returns <b>a</b>:</p> <pre>var xml = XML.parse('&lt;a b="1"&gt;A&lt;/a&gt;'); return xml.name;</pre>                                                                                                                                |
| <b>data</b>              | <p>Returns the data of an element, when the element is a data node. The text function is recommended rather than the data function.</p> <p>The following example returns <b>A</b>:</p> <pre>var xml = XML.parse('&lt;a b="1"&gt;A&lt;/a&gt;'); return xml.children[0].data;</pre>                   |
| <b>text</b>              | <p>Returns the text of an element. Checks whether the node is a data node or has a data node as a child.</p> <p>The following example returns <b>A</b>:</p> <pre>var xml = XML.parse('&lt;a b="1"&gt;A&lt;/a&gt;'); return xml.text;</pre>                                                          |
| <b>getAttribute(str)</b> | <p>Returns an attribute value.</p> <p>The following example returns <b>1</b>.</p> <pre>var xml = XML.parse('&lt;items&gt;&lt;item att="1"&gt;A&lt;/item&gt;&lt;/items&gt;'); var item = xml.query('item'); var att = item.getAttribute('att'); return att;</pre>                                    |
| <b>remove</b>            |                                                                                                                                                                                                                                                                                                     |

## Table Functions

The following functions affect tables in the Dataflow Editor.

|                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>tableGetRows(table)</b>                              | <p>Gets a list of rows from a table object. Uses Dataflow Editor syntax to find the table.</p> <p>For example, for the default table with sample data, the following function returns <code>[[0,1,4,"v"],[1,3,2,"a"],[2,2,6,"k"]]</code>:</p> <pre>tableGetRows(@parent.table)</pre>                                                                                                                                                          |
| <b>tableGetColumns(table)</b>                           | <p>Gets information about each column in a table object. Uses Dataflow Editor syntax to find the table.</p> <p>For example, for the default Table block sample data, the function <code>tableGetColumns(@parent.table)</code> returns the following result:</p> <p><b>[&lt;first column data&gt;,{" name":"v1"," type":" string"," width":100," formula": null," meta":{" name":"v1"}},&lt;column v2 data&gt;,&lt;column v3 data&gt;]</b></p> |
| <b>tableSet(table, rows, [columns], serialize=true)</b> | <p>Sets data in the specified table. Uses Dataflow Editor syntax to find the table.</p> <p>For example, the following function creates the table shown in Figure 189.</p> <pre>tableSet(@parent.table, [{"a","b"}, {"c","d"}], ["col1", "col2"]);</pre>                                                                                                                                                                                       |
| <b>tableAddRows(table, rows)</b>                        | <p>Add rows to a table object. A dummy value must be passed as the first item in any row. This dummy value will always be overridden by the automatically generated row ID in the <b>row</b> column.</p> <p>For example, the following function adds two rows to a table:</p> <pre>tableAddRows(@parent.table, [{"", "a", "b"}, [{"", "c", "d"}]);</pre>                                                                                      |
| <b>tableRemoveRows(table, start, [count=1])</b>         | <p>Removes a number of rows from a table object.</p> <p>For example, the following function removes two rows from a table, beginning with the row at index 0:</p> <pre>tableRemoveRows(@parent.table, 0, 2);</pre>                                                                                                                                                                                                                            |
| <b>tableClear(table)</b>                                | <p>Clears the table. Uses Dataflow Editor syntax to find the table.</p> <p>The following example clears all rows from a table. Columns remain.</p> <pre>tableClear(@parent.table);</pre>                                                                                                                                                                                                                                                      |
| <b>\$thisRow['column Name']</b>                         | <p>Selects the column value from the current row being iterated on by a Filter or Column Mapping block. The following is an example:</p> <pre>v1+\$thisRow['v2']</pre>                                                                                                                                                                                                                                                                        |

**Kinetic - Edge & Fog Fabric Processing Module**  
**Dataflow Editor User Guide**

---

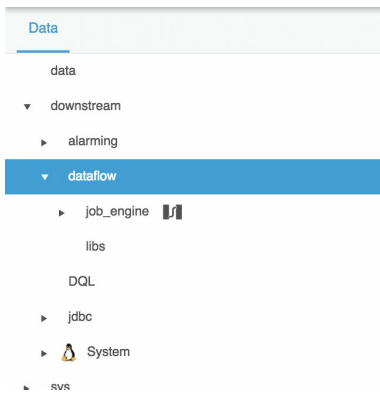
Appendixes

**Figure 189. Result of tableSet Function Example**

| row | col1 | col2 |
|-----|------|------|
| 0   | a    | b    |
| 1   | c    | d    |

### Use of the dataflow/libs path

The share libs folder allows for storing and sharing symbols across dataflows. This is done by taking any dataflow containing only symbols and placing it into the downstream/libs folder. In this manner, for any new dataflow, the user can drag the dataflow in the libs folder to the “symbols” tab of the dataflow. That will have the effect of making those symbols available in your new dataflow. Basically, it implements a way of sharing the same symbol implementation across multiple dataflows without having to export/import them all the time.



## Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

<http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html>

Subscribe to the *What's New in Cisco Product Documentation* as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS Version 2.0.

---

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: [www.cisco.com/go/trademarks](http://www.cisco.com/go/trademarks). Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.