Tag Historian

The Tag Historian module provides power and flexibility for storing and accessing historical data. When history is enabled on an Ignition Tag, data is stored automatically in your SQL database in an efficient format. This data is then available for querying through scripting, historical bindings, and reporting. Options for partitioning and deleting old data help to ensure the system stays maintained with minimal extra work. Also, you can drag-and-drop Tags directly onto an Easy Chart component to create trends or onto a table to display historical values. Tag Historian's robust querying features provide you great flexibility in how you retrieve the stored data.

On this page ...

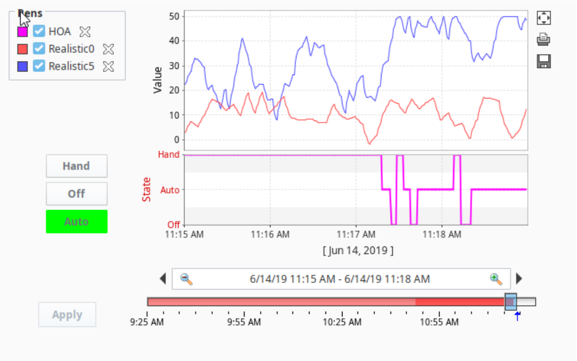
Tag Historian Querying

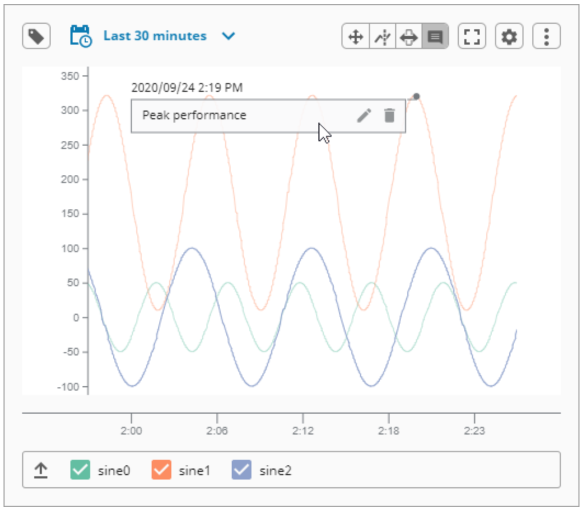
While the data is stored openly in the database, it does not lend itself well to direct querying. Ignition offers a range of built-in querying options that are very powerful and flexible. In addition to simple on-change querying, the system can perform advanced functions such as querying many Tags from multiple providers, calculating their quality, interpolating their values, and coordinating their timestamps to provide fixed resolution returns. Tag history bindings allow you to pull Tag history data that is stored in the database into a component through a binding. The binding type, which is only available for Dataset type properties, runs a query against the Tag Historian.
For more information, see Tag History Bindings in Perspective or Tag History Bindings in Vision.

Querying can be performed on tables and charts through the Historical binding, Nested Queries, and through scripting. You can also query Tags from the Reporting module.

Charts that Display Historian Data

With the Vision module Easy Chart component you can create powerful timeseries charts from Tag Historian data. As of 8.1.0, the Perspective module has a Power Chart component with similar capabilities. Both components enable you to drag and drop history-enabled Tags onto a chart to create chart pens and data trends. Your charts and graphs can include subplots, axes, digital offsets, and moving averages. You can quickly and easily turn your historical and realtime data into dynamic charts and graphs for your users. These charts can be configured in the runtime to give users quick access to data in the time range they need.





Store and Forward

The Store and Forward system provides a reliable way for Ignition to store historical data to the database. The Store and Forward system is not exclusively part of Tag history, but systems such as the Tag Historian and Transaction Groups use it to prevent data loss and store data efficiently using a record cache.

Data storage

Historical Tag values pass through the Store and Forward system before they are stored in the database connection associated with the historian provider. The data is stored according to its data type directly to a table in the SQL database, with its quality and a millisecond resolution timestamp. The data is only stored on-change, according to the value mode and deadband settings on each Tag, thereby avoiding duplicate and unnecessary data storage. The storage of scan class execution statistics ensures the integrity of the data. While advanced users may change the table according to their database to be more efficient (for example, using a compression engine), Ignition does not perform binary compression or encrypt the data.

Configuring Tag History

This feature was changed in Ignition version **8.1.17**:

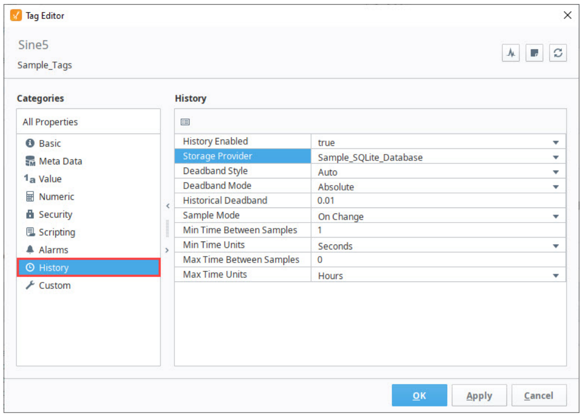
In 8.1.17, the Tag Editor was redesigned to improve usability. The new Tag Editor now requires fewer clicks and keeps relevant tag information visible while modifying bindings, alarms, and event scripts.

Pages detailing features of the previous Tag Editor can be found in Deprecated Ignition Features.

Logging data is easy with Tag Historian. Once you have a database connection, all you do is set the tags to store history and Ignition takes care of the work. Ignition creates the tables, logs the data, and maintains the database.

The historical tag values pass through the store-and-forward engine before ultimately being stored in the database connection associated with the historian provider. The data is stored according to its data type directly to a table in the SQL database, along with its quality and a millisecond resolution timestamp. The data is only stored on-change, according to the value mode and deadband settings on each tag, thereby avoiding duplicate and unnecessary data storage. The storage of Tag Group execution statistics ensures the integrity of the data.

The first step to storing historical data is to configure tags to record values. This is done from the History section of the Tag Editor in the Designer. Select the **History Enabled** property to turn on history. The properties include an Historical Tag Group that will be used to check for new values. Once values surpass the specified deadband, they are reported to the history system, which then places them in the proper store and forward engine.



History Enabled	true
Storage Provider	Sample_SQLite_Database
Deadband Style	Auto
Deadband Mode	Absolute
Historical Deadband	0.01
Sample Mode	On Change
Min Time Between Samples	1
Min Time Units	Seconds
Max Time Between Samples	0
Max Time Units	Hours

On this page ...

INDUCTIVE UNIVERSIT

Configuring Tag History

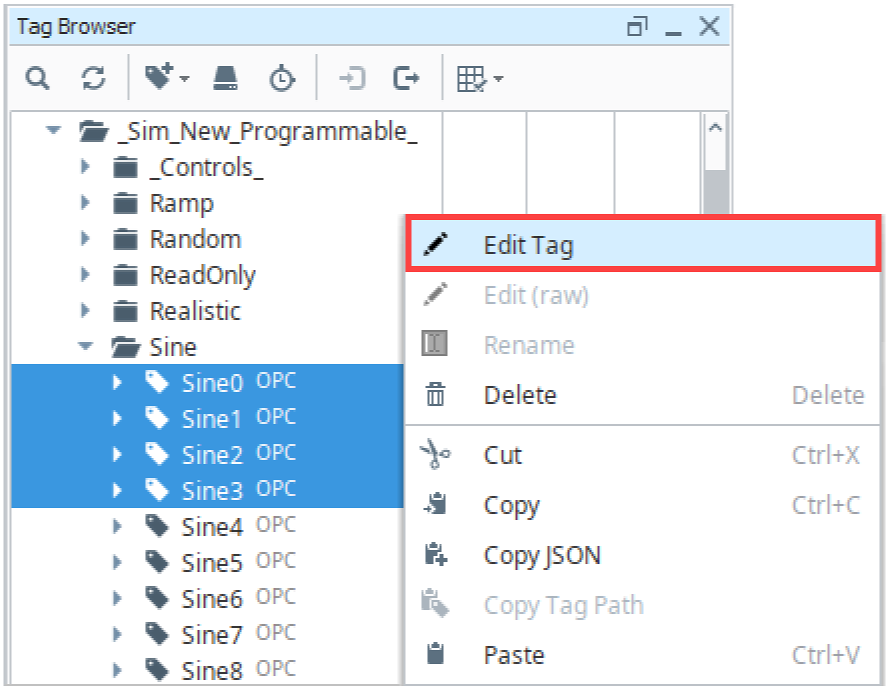
Watch the Video

Enable History on a Tag

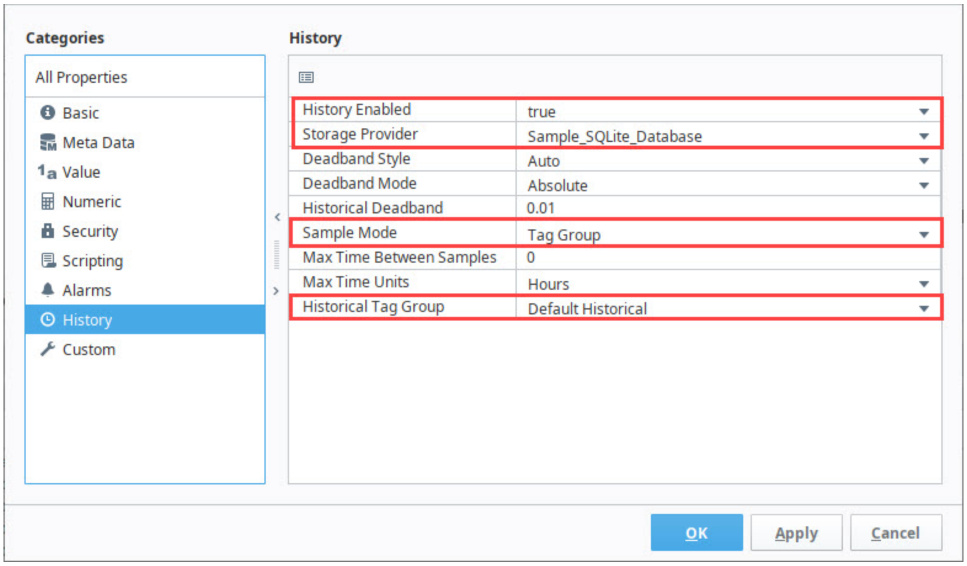
The following example demonstrates how to configure a tag to store values into the Tag Historian. Complete information on the History properties (and all properties in the Tag Editor), can be found on the Tag Properties Table.

Note: Dataset type tags are not supported by the Tag History system.

1. In the Tag Browser, select one or more tags. For example, we selected several **Sine** tags in the Sine folder.
2. Right-click on the selected tags, and then select **Edit Tag** ✎ .
   The Tag Editor window is displayed where you can change the tag name, data type, scaling options, metadata, permissions, history, and alarming.

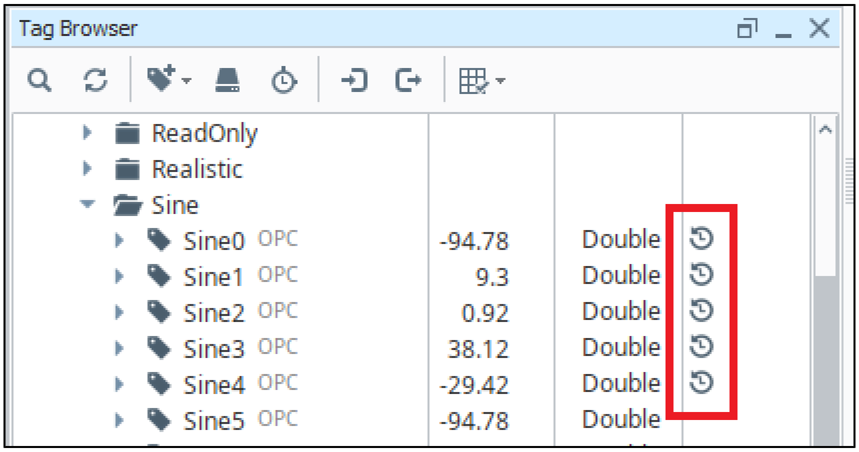


3. Scroll down to History or select the History pane on the Tag Editor. Set **History Enabled** to true.
4. Choose a database (for example, SQLite) from the **Storage Provider** dropdown.
5. Set the Sample Mode to **Tag Group**.
6. Set the Historical Tag Group to **Default Historical**.



History Enabled	true
Storage Provider	Sample_SQLite_Database
Deadband Style	Auto
Deadband Mode	Absolute
Historical Deadband	0.01
Sample Mode	Tag Group
Max Time Between Samples	0
Max Time Units	Hours
Historical Tag Group	Default Historical

7. Click **OK**. Now look in the Tag Browser. To the right of each Sine tag that is storing history, a **History** ⟲ icon appears letting you know it is set up.



ReadOnly			
Realistic			
Sine			
Sine0 OPC	-94.78	Double	⟲
Sine1 OPC	9.3	Double	⟲
Sine2 OPC	0.92	Double	⟲
Sine3 OPC	38.12	Double	⟲
Sine4 OPC	-29.42	Double	⟲
Sine5 OPC	-94.78	Double	

Now, if you look in your database, you can see all the tables and data Ignition has created for you.

Setting a UDT to Log History Data

Enabling tag history on members in a UDT involves editing the UDT definition, and enabling history on the any members you wish to record. These history settings will then propagate to members in any instances.

Tag History Configuration

Below is a description of some important tag history settings.

Sample Mode

The Sample Mode setting determines how often the Tag History system will check if the tag's value should be stored to the database using Deadband and Deadband Style settings.

- **On Change** - Ignition will check if the value should be stored to the database each time the tag value changes.
- **Periodic** - Ignition will check if the value should be stored to the database at a specific rate defined by the **Sample Rate** property.
- **Tag Group** - Ignition will check if the value should be stored to the database at the specific rate defined within the Tag Group selected under the **Historical Tag Group** property.

Historical Tag Group

The Historical Tag Group setting is visible when Sample Mode is set to Tag Group. The Historical Tag Group setting determines how often to check if the value on the tag should be stored. It uses the same Tag Groups that dictate how often your tags should execute. Typically, the Historical Tag Group should execute at the same rate as the tag's Tag Group or slower. For example, if a tag's Tag Group is set to update at a 1,000ms rate, but the Historical Tag Group is set to a Tag Group that runs at 500ms rate, then the Tag History system will be checking the tag's value twice between normal value changes, which is unnecessary.

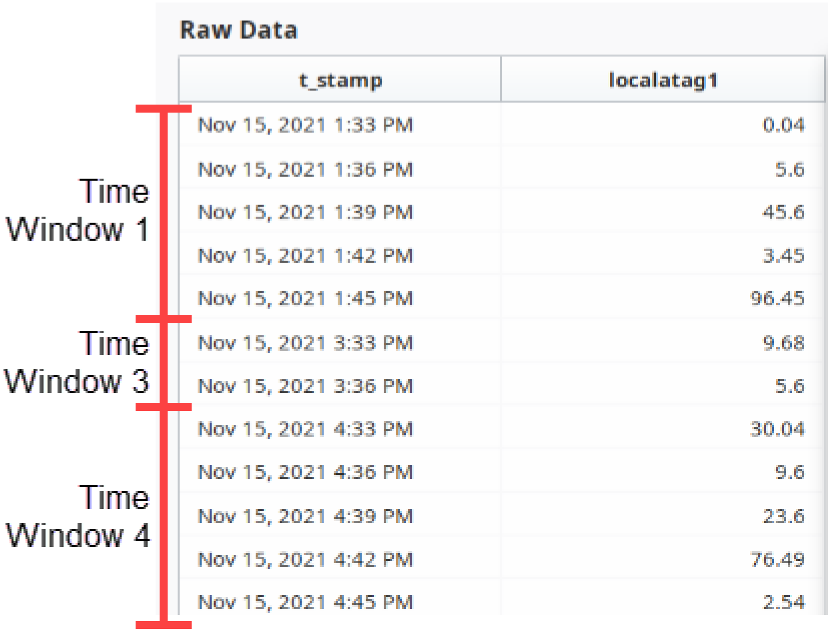
Max and Min Time Between Samples

Normally, Tag Historian only stores records when values change. By default, an "unlimited" amount of time can pass between records – if the value doesn't change, a new row is never inserted in the database. By modifying these settings, it is possible to specify the maximum number of Tag Group execution cycles that can occur before a value is recorded. Setting the value to 1, for example, would cause the tag value to be inserted each execution, even if it has not changed. Given the amount of extra data in the database that this would lead to, it's important to only change this property when necessary.

Interpolated Values

When the Ignition historian queries a database for historical data, there may be intervals with no raw data in the database. When this happens, Ignition interpolates the missing data. Interpolation is the process of estimating unknown values that fall between known values, and is calculated depending on the tag configuration or selected **Aggregation Mode** , as long as **AvoidInterpolation** is false. Although interpolated values are calculated every time a raw data point is found or when an interval ends based on the configuration or Aggregation Mode selections, the values are reserved to be used only when there are no data points in an interval.

To demonstrate how interpolation works, example data queried using a traditional Tag History binding from 1:30pm to 5:30pm in 60 minute time windows is shown below. The query returned three windows with data and one window with no data.



Raw Data

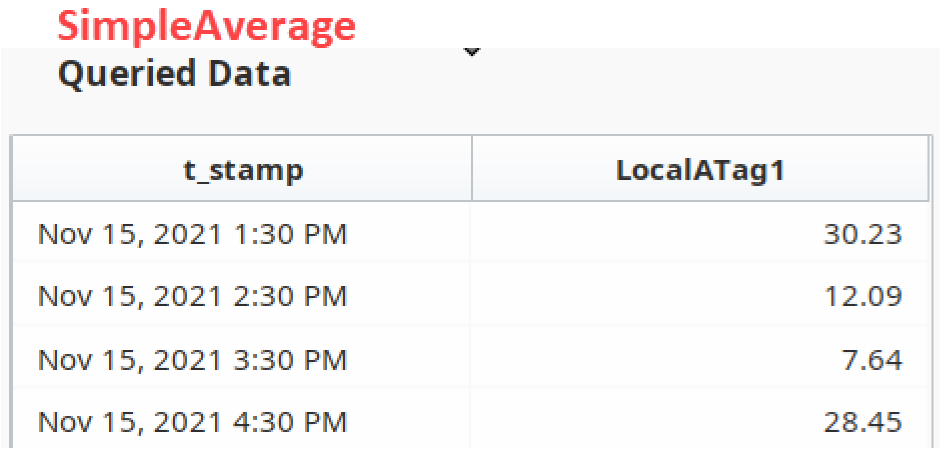
t_stamp	localatag1
Nov 15, 2021 1:33 PM	0.04
Nov 15, 2021 1:36 PM	5.6
Nov 15, 2021 1:39 PM	45.6
Nov 15, 2021 1:42 PM	3.45
Nov 15, 2021 1:45 PM	96.45
Nov 15, 2021 3:33 PM	9.68
Nov 15, 2021 3:36 PM	5.6
Nov 15, 2021 4:33 PM	30.04
Nov 15, 2021 4:36 PM	9.6
Nov 15, 2021 4:39 PM	23.6
Nov 15, 2021 4:42 PM	76.49
Nov 15, 2021 4:45 PM	2.54

- **Time Window 1** - 1:30pm through 2:30pm: 0.04, 5.6, 45.6, 3.45, 96.45
- **Time Window 2** - 2:30pm through 3:30pm: no data in this time window
- **Time Window 3** - 3:30pm through 4:40pm: 9.68, 5.6
- **Time Window 4** - 4:30pm through 5:30pm: 30.04, 9.6, 23.6, 76.49, 2.54

If you select **SimpleAverage** as your Aggregation Mode , the request returns a row representing the **SimpleAverage** value for every time window. The value in the first row was calculated by averaging all of the values between 1:30pm - 2:30pm:

```
(0.04 + 5.6 + 45.6 + 3.45 + 96.45) / 5 = 30.23
```

This same method applies to the third and fourth row values. However, since there are no raw records between 2:30pm - 3:30pm, the value of 12.09 is calculated using interpolation.

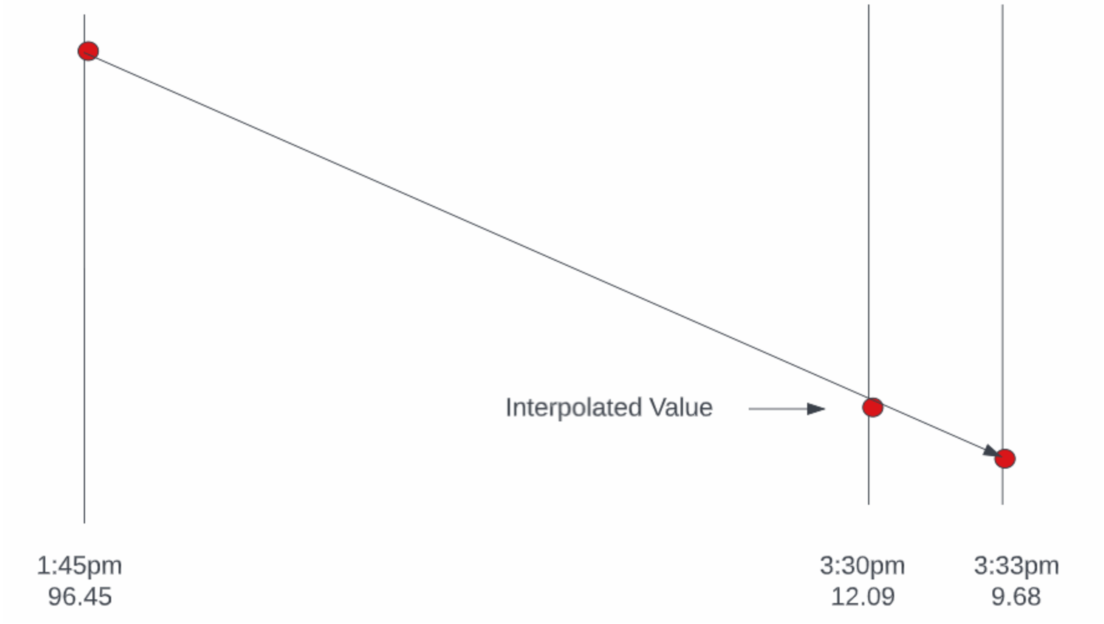


SimpleAverage

Queried Data

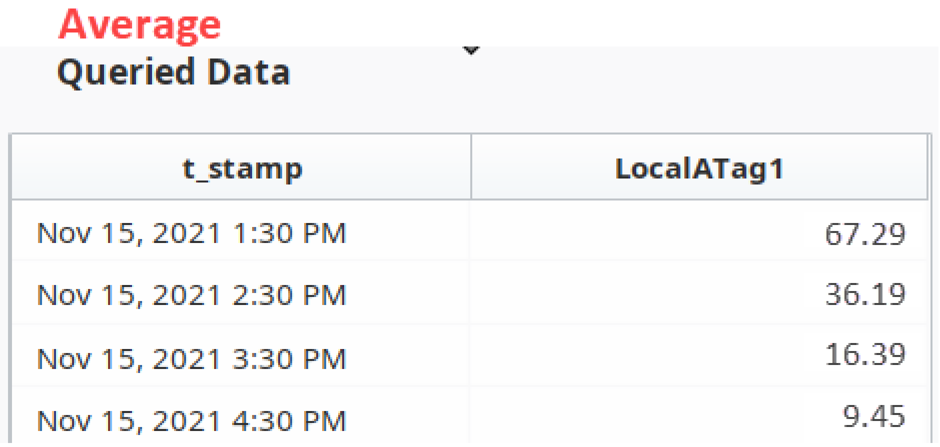
t_stamp	LocalATag1
Nov 15, 2021 1:30 PM	30.23
Nov 15, 2021 2:30 PM	12.09
Nov 15, 2021 3:30 PM	7.64
Nov 15, 2021 4:30 PM	28.45

12.09 was obtained by marking the end point of the empty window along a line connecting the last value from the previous window to the first value in the next window. This is easily shown using a chart to visualize where the time and value intersect. The last value in the previous time window is marked at 96.45 and the first value in the next time window is marked at 9.68. Once the line is drawn between the two, the marked value at the end of the empty window end time (3:30pm) gives us the interpolated value of 12.09.



The interpolated value remains 12.09 for every Aggregation Mode we query for unless we select the **Average** Aggregation Mode, which returns a value of 36.19.

The **Average** Aggregation Mode is an exception to the previously described interpolation method because the time-weighted calculation included in the **Average** process remains the same whether there are or aren't data points in a time interval.



Queried Data

t_stamp	LocalATag1
Nov 15, 2021 1:30 PM	67.29
Nov 15, 2021 2:30 PM	36.19
Nov 15, 2021 3:30 PM	16.39
Nov 15, 2021 4:30 PM	9.45

The time-weighted process uses current and previous values with the interval time difference in milliseconds to determine **Average** values.

- **Average** Value = (0.5 * abs((currVal - prevVal) * timediff)) + (timediff)(min(currVal,prevVal))

If we use the same data from the earlier example, the process looks like:

```
prevVal = 96.45
currVal = 60.29
```

Since the current value, or end time of the first window, has no data the value is obtained with a **retValue** process. The required data for this process includes the last value in the previous non-empty window, represented by A; the first value in the next non-empty window, represented by B; and the amount of time between the last value in the previous non-empty window divided by the total time between collected values.

- retValue = A + ((B-A) * ((ts - Ats) / (Bts - Ats))

```
A = 96.45
B = 9.68
Bts - Ats = 3:33pm - 1:45pm > 108 minutes
ts - Ats = 2:30pm - 1:45pm > 45 minutes

96.45 + ((9.68 - 96.45) (45/108)) = 60.29
```

If there was data for the 2:30pm interval, which is the end of Time Window 1, that would be used as the current value and no further calculations would be required.

Since the previous value was collected at 1:45pm and the current value was collected at 2:30pm, the time difference is 45 minutes. Since time difference is calculated in milliseconds, 2,699,999 is used:

```
timediff = 2,699,999
```

Therefore the process calculation for the first window is:

```
(0.5 * abs((60.2958 - 96.45) * 2,699,999)) + (2,699,999)(min(60.2958)) = 211,606,751.6271
```

Add this result to the sum of all **Average** values for the data between 1:33pm and 1:45pm to include all time-weighted data.

The same process calculation is used for the values between 1:33pm and 1:45 as used for the first window calculation above.

```
Time                Value
1:33                0.04
1:36                5.6
1:39                45.6
1:42                3.45
1:45                96.45

1:33 to 1:36:
(0.5 * abs((5.6-0.04) * 179,999)) + 179,999(0.04) = 507,597.18

1:36 to 1:39:
(0.5 * abs((45.6-5.6) * 179,999)) + 179,999(5.6) = 4,607,974.4

1:39 to 1:42:
(0.5 * abs((3.45-45.6) * 179,999)) + 179,999(3.45) = 4,414,475.47

1:42 to 1:45:
(0.5 * abs((96.45-3.45) * 179,999)) + 179,999(3.45) = 8,990,950.05

507,597.18 + 4,607,974.4 + 4,414,475.47 + 8,990,950.05 = 18,520,997.1
```

Then, divide by the total time that has passed, which is 57 minutes (1:33pm-2:30pm) or 3,419,999ms:

```
(211,606,751.6271 + 18,520,997.1) / 3,419,999 = 67.29
```

As shown in the queried values above, 67.29 is the **Average** value for Time Window 1.

Now, instead of just creating a chart to find the linear interpolated value and moving on, we must follow the same **Average** process for the empty window to apply the time-weighted calculations. To do this, we will still use the linear interpolated value of 12.09 as the current value in the process since the window has no data. The current value of 60.29 used in the Time Window 1 process becomes the previous value for this Time Window 2 process.

```
prevVal = 60.29
currVal = 12.09
```

Since the time window interval is from 2:30pm to 3:30pm, we use 60 minutes, or 3599999ms for the time difference:

```
timediff = 3599999
```

Therefore, the process calculation for the empty window is:

```
(0.5 * abs((12.09 - 60.2958) * 3,599,999)) + (3,599,999)(min(12.09)) = 130,283,963.81
```

Lastly, to get the **Average** interpolated value this number is divided by the total time that has passed, which is 60 minutes or 3,599,999ms:

```
130,283,963.81 / 3,599,999 = 36.19
```

Deadband and Analog Compression

The deadband value is used differently depending on whether the tag is configured as a Discrete Tag or as an Analog Tag. Its use with discrete values is straightforward, registering a change any time the value moves +/- the specified amount from the last stored value. With Analog Tags, however, the deadband value is used more as a compression threshold, in an algorithm similar to that employed in other Historian packages. It is a modified version of the 'Sliding Window' algorithm. Its behavior may not be immediately clear, so the following images show the process in action, comparing a raw value trend to a "compressed" trend.

The Deadband Style property sets the: Auto, Analog, or Discrete.

Discrete

Storage

The deadband will be applied directly to the value. That is, a new value (V_1) will only be stored when: $|V_1-V_0|$ >= Deadband.

Interpolation

The value will not be interpolated. The value returned will be the previous known value, up until the point at which the next value was recorded.

Analog

Storage

Every time the tag's value changes, this method will calculate upper and lower slope values. These slope values are stored in memory, and are ultimately used to determine when a new value is stored. The calculations used are listed below:

Upper Slope

```
(((NewValue + Deadband) - PreviousValue) / (NewTimestamp - PreviousTimestamp))
```

Lower Slope

```
(((NewValue - Deadband) - PreviousValue) / (NewTimestamp - PreviousTimestamp))
```

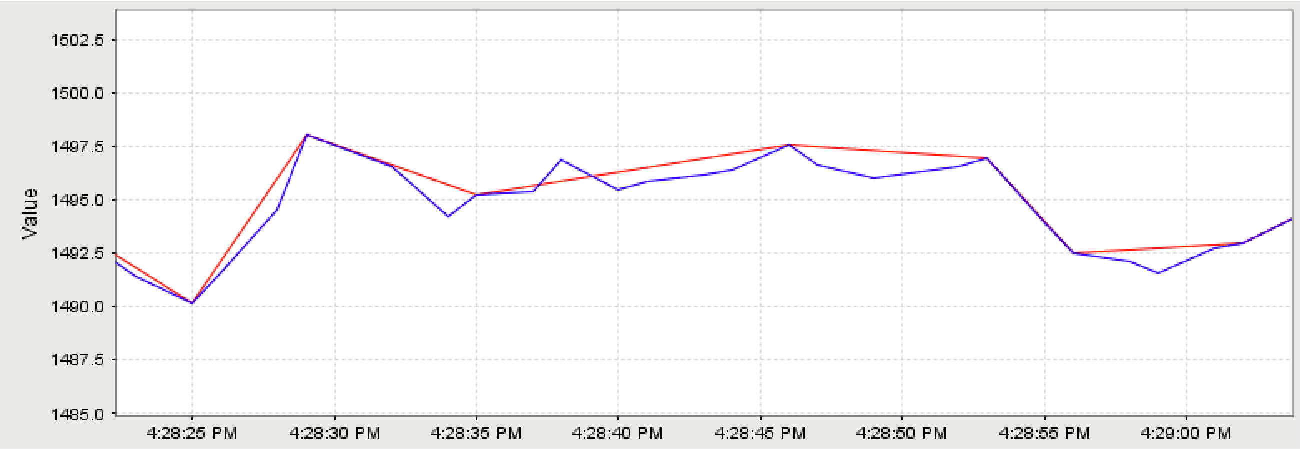
The algorithm will only store new values under the following conditions:

- The system always stores the first value on the tag when using the method, since the subsequent values will need an initial value to calculate slope from.
- If the newly calculated upper slope is lower than the previously calculated lower slope value, the system will store the new value.
- If the newly calculated lower slope is larger than the previously calculated upper slope value, the system will store the new value.
- The system always stores a value when the quality on the tag changes.

In cases where a new value isn't stored, the system will compare the newly calculated slope values to the previously calculated values:

- If the new upper slope is less than the previous upper slope, then the new upper slope is used for future comparisons.
- If the new lower slope is greater than the previous lower slope, then the new lower slope is used for future comparisons.

In the image below, an analog value has been stored. The graph has been zoomed in to show detail; the value changes often and ranges over time +/- 10 points from around 1490.0. The compressed value was stored using a deadband value of 1.0, which is only about .06% of the raw value, or about 5% of the effective range. The raw value was stored using the Analog mode, but with a deadband of 0.0. While not exactly pertinent to the explanation of the algorithm, it is worth noting that the data size of the compressed value, in this instance, was 54% less than that of the raw value.

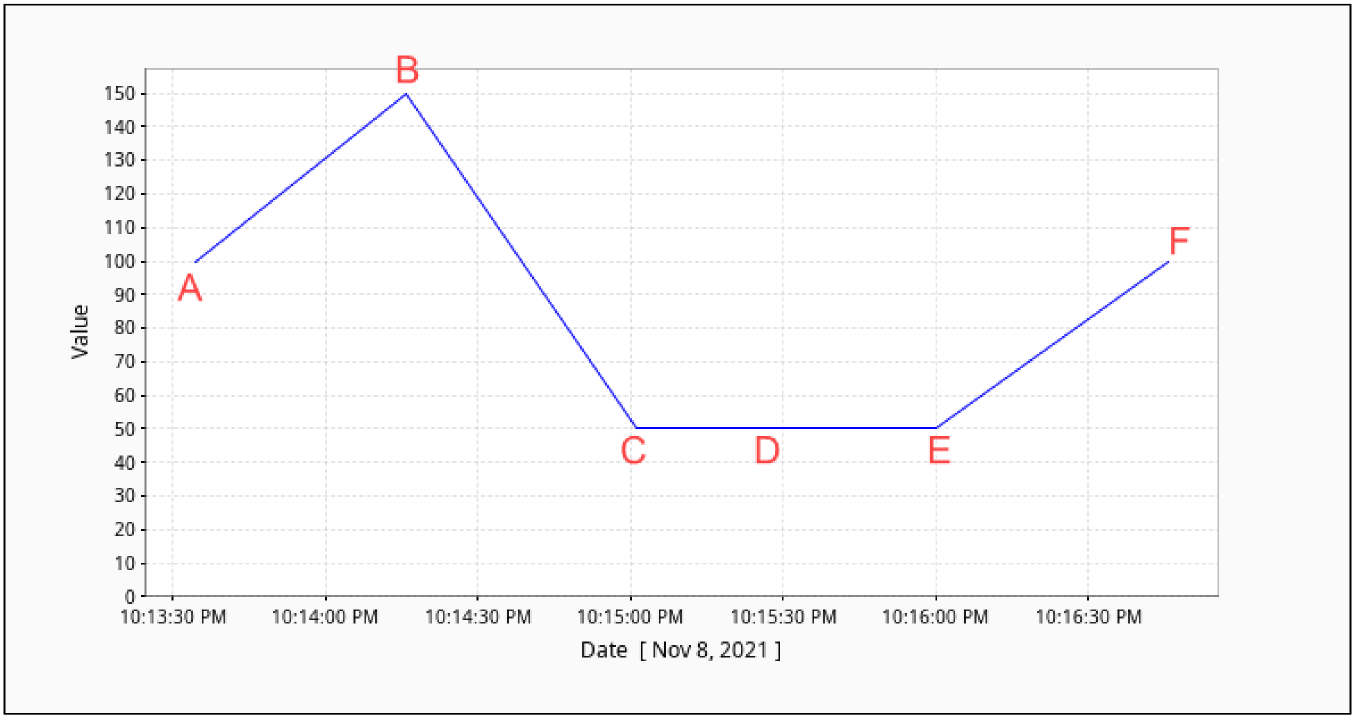


Interpolation

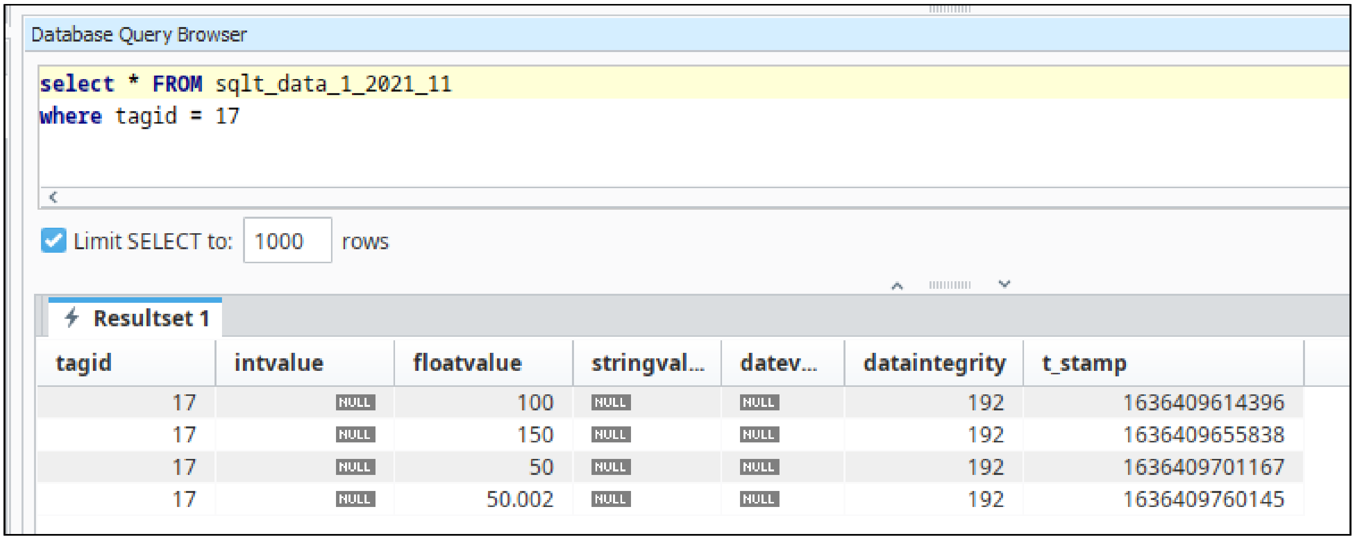
The value will be interpolated linearly between the last stored value and the next value. For example, if the value at Time$_0$ was 1, and the value at Time$_2$ is 3, selecting Time$_1$ will return 2.

Example

Let's look at a demonstration of how the analog compression works. For this example we'll assume a tag is using a historical deadband value of 0.01. Over the course of a few moments the tag's value changed several times, as represented on the chart below.



The tag historian system stores the records into one of the data partitions. After the value changes above, our database stores the records.



```
select * FROM sqlt_data_1_2021_11
where tagid = 17
```

☑ Limit SELECT to: 1000 rows

tagid	intvalue	floatvalue	stringval...	datev...	dataintegrity	t_stamp
17	NULL	100	NULL	NULL	192	1636409614396
17	NULL	150	NULL	NULL	192	1636409655838
17	NULL	50	NULL	NULL	192	1636409701167
17	NULL	50.002	NULL	NULL	192	1636409760145

Below we'll describe how each value was stored as the tag changed value.

Value A

Once we enable history on the tag and set the deadband style to Analog, the system will record the first value on the tag. Since we only have our first value, we use arbitrarily large and small values for the upper slope and lower slope (3.40282347 x 10^38 and -3.40282347 x 10^38, respectively), and store those numbers until the tag changes value again.

Value B

Here we see the value on the tag changed to 150. Since this is only the second value recorded, the system needs to figure out the slope values so it knows when to next collect a record. The system calculates both slope values as listed above.

```
// Upper Slope
((150 + 0.01) - 100) / (1636409655838 - 1636409614396)
(150.01 - 100) / 41442
50.01 / 41442
0.001206768

// Lower Slope
((150 - 0.01) - 100) / (1636409655838 - 1636409614396)
(149.99 - 100) / 41442
49.99 / 41442
0.0012062641
```

Because the previously stored slope values are simply placeholders, we replace them with these newly calculated values. This value of 150 is **not yet** stored in the database. Instead, this value of 150 is kept in memory, waiting until the tag changes again.

Value C

Our tag changes value to 50. The system calculates the new slope values again, this time using 150 as the previous value.

```
// Upper Slope
((50 + 0.01) - 150) / (1636409701167 - 1636409655838)
(50.01 - 150) / 45329
-99.99 / 45329
-0.0022058727

// Lower Slope
((50 - 0.01) - 150) / (1636409701167 - 1636409655838)
(49.99 - 150) / 45329
-100.01 / 45329
-0.002206314
```

Our newly calculated values meet our storage criteria. The new upper slope (-0.0022058727) is less than the previous lower slope (0.0012062641). Therefore the system will store the previous value (150) and use these newly calculated slope values the next time the tag changes value. The newest value of 50 is not yet stored.

Value D

Our tag changes to a value of 50.001. As usual, the system calculates some new slope values.

```
// Upper Slope
((50.001 + 0.01) - 50) / (1636409726809 - 1636409701167)
(50.011 - 50) / 25642
0.011 / 25642
4.289837E-7

// Lower Slope
((50.001 - 0.01) - 50) / (1636409726809 - 1636409701167)
(49.991 - 50) / 25642
-0.009 / 25642
-3.5098665E-7
```

Our new lower slope is larger than our previously stored upper slope. We record the previous value of 50 and keep our new slope values in memory.

Value E

Our tag changes to a value of 50.002. We calculate new slope values.

```
// Upper Slope
((50.002 + 0.01) - 50) / (1636409760145 - 1636409701167)
(50.012 - 50) / 58978
0.012 / 58978
2.034657E-7

// Lower Slope
((50.002 - 0.01) - 50) / (1636409760145 - 1636409701167)
(49.992 - 50) / 58978
-0.008 / 58978
-1.356438E-7
```

These new slope values do **not** meet our storage criteria: the new upper slope is not less than the previous lower slope, and the new lower slope isn't greater than the previous upper slope. Thus, the previous tag value of 50.001 is not stored since it's too similar to the current value of 50.002.

In addition, the system does notice that the new upper slope is less than the old upper slope, and the new lower slope is greater than the old lower slope. So the system deems the new slope values to be more restrictive, and will use those the next time the tag value changes. The system will use the newly calculated slope values when evaluating the next value change.

Value F

Our tag changes to a value of 100. New slope values are calculated, using the most recent value of 100 compared to the previous value of 50.002.

```
// Upper Slope
((100 + 0.01) - 50.002) / (1636409786810 - 1636409760145)
(100.01 - 50.002) / 26665
50.008 / 26665
0.0018754172

// Lower Slope
((100 - 0.01) - 50.002) / (1636409786810 - 1636409760145)
(99.99 - 50.002) / 26665
49.988 / 26665
0.0018746671
```

The new lower slope is greater than the previous lower slope, so the previous value (50.002) is stored. This process repeats indefinitely.

Auto

The setting will automatically pick either Analog or Discrete, based on the data type of the tag.

- If the data type of the tag is set to a float or double, then Auto will use the Analog Style.
- If the data type of the tag is any other type, then the Discrete style will be used.

Seeded Values

Tag history queries sometimes use seeded values (occasionally called "Boundary Values"). When retrieving tag history data, the system will also retrieve values just outside of the query range (before the start time, after the end time), and include them in the returned result set. They're generally used for interpolation purposes. If you do not want to include these seeded values, interpolation must not be enabled. The following is also required:

- Have the tag store history with a Discrete Mode
- Set the [noInterpolation] parameter to `true`
- Set the [includeBoundingValues] parameter to `false` on the calling query

Pre-Query Seed Value

These are a single value taken from just before the start of the query range. The value and timestamp for this value is typically the first row in the resulting query. Pre-query seed values are always included when not using a raw data query.

An exception to this rule is can be found with the system.tag.queryTagHistory function. Setting `includingBoundingValues` argument to True and `returnSize` to -1 will return a raw data query with a pre-query seed value.

Post-Query Seed Value

These extra values are added to the end of the result set, representing the next data point after the query range. Post-query seed values are only included when interpolation is requested/enabled for the query. Thus, values stored with a Discrete deadband style will not include post-query seed values in the query results, but an Analog deadband style will include post-query seed values.

If the system knows the query is retrieving records for a tag on the local system, this value will be determined by the current tag's value instead of retrieving the last recorded value in the database. The current tag's value is also used in cases where the time range extends to the present time.

> **Note:** If a result in a query is outside of the requested range, the value is typically a seeded value. This typically occurs when the range is so small that values were not recorded, or when the range is in the future (and thus values have not yet been recorded).

Raw Data Queries

In most cases queries returned by tag history will apply some form of aggregation. However it is possible to get a "raw data query", which is a result set that contains only values that were recorded: meaning no aggregation or interpolation is applied to the results. A raw data query can be obtained by using one of the following options:

- Set the Sample Size on Vision Tag History bindings to **On Change**
- Setting the **returnSize** parameter on system.tag.queryTagHistory or system.tag.queryTagCalculations to -1.

  > **Note:** Be aware that if a tag is storing history using the Analog style, the returned dataset will include post-query seed values.

- Settting the Query Mode on Perspective Tag History bindings to **AsStored**

Custom Tag History Aggregates

Python Aggregation Functions

The Tag History system has many built-in aggregate function, such as Average, Sum, and Count. However a custom aggregate may be defined via Python scripting. These functions are used for calculations across timeframes, and they process multiple values in a “window” into a single result value.

For example, if a query defines a single row result, but covers an hour of time (either by requesting a single row, or using the Tag Calculations feature), the system must decide how to combine the values. There are many built in functions, such as Average, Sum, Count, etc. Using a custom Python aggregate, however, allows you to extend these functions and perform any type of calculation.

On this page ...

Description

As values come in, they will be delivered to this function. The interpolator will create and deliver values. For each window (or “data block”, the terms are used synonymously), the function will get a fresh copy of blockContext. The blockContext is a dictionary that can be used to as a memory space. The function should not use global variables. If values must be persisted across blocks, they can be stored in the queryContext, which is also a dictionary.

The function can choose what data to include, such as allowing interpolation or not, and allowing bad quality or not.

The window will receive the following values, many of which are generally interpolated (unless a raw value happens to fall exactly at the time):

- The start of the window
- 1 ms before each raw value (due to the difference between discrete and analog interpolation. A value equal to the previous raw value indicates discrete interpolation)
- The raw value
- The end of the window

At the end of the window, the function will be called with “finished=true”. The function should return the calculated value(s). The resulting value will have a timestamp that corresponds to the beginning of the block timeframe.

Parameters

- qval - The incoming QualifiedValue. This has:
  - value : Object
  - quality : Quality (which has ‘name’, ‘isGood()’)
  - timestamp : Date
- interpolated - Boolean indicating if the value is interpolated (true) or raw (false)
- finished - Boolean indicating that the window is finished. If true, the return of this particular call is what will be used for the results. If false, the return will be ignored.
- blockContext - A dictionary created fresh for this particular window. The function may use this as temporary storage for calculations. This object also has:
  - blockId - Integer roughly indicating the row id (doesn’t take into account aggregates that return multiple rows)
  - blockStart - Long UTC time of the start of the window
  - blockEnd - Long UTC time of the end of the window
  - previousRawValue - QualifiedValue, the previous non-interpolated value received before this window
  - previousBlockResults - QualifiedValue[], the results of the previous window.
  - insideBlock(long) - Returns boolean indicating if the time is covered by this window.
  - get(key, default) - A helper function that conforms to python’s dictionary “get with default return”.
- queryContext - A dictionary that is shared by all windows in a query. It also has:
  - queryId - String, an id that can be used to identify this query in logging
  - blockSize - Long, time in ms covered by each window
  - queryStart - Long, the start time of the query
  - queryEnd - Long, the end time of the query
  - logTrace(), logDebug(), logInfo() - all take (formatString, Object... args).

Return Value

Your custom aggregate should return one of the following for each window:

- Object - Turned into Good Quality qualified value
- List - Used to return up to 2 values per window
- Tuple - (value, quality_int)
- List of quality tuples

Usage

Custom Python aggregates can be used in two ways:

1. Defined as a Project Library script, where the full path to the function is passed to the query.
2. Defined as a string, prefaced with “python:”, and passed to the query.

Currently both options are only available through the system.tag.queryTagHistory/queryTagCalculations functions.

Both of these options are used with the “aggregationMode” and “ aggregationModes” parameters to system.tag.queryTagHistory, and the “calculations” parameter of system.tag.queryTagCalculations. If the value is not an Enum value from the defined AggregationModes, it will be assumed to be a custom aggregate. The system will first see if it’s the path to a Project Library script, and if not, will then try to compile it as a full function.

For performance reasons, it is generally recommended to use the Project Library script whenever possible. For more information, see Project Library.

Conditions

There are some key factors to keep in mind when calling a custom aggregate.

Return Size

When calling a custom tag history aggregate, the `returnSize` argument must be set to a number greater than 0, otherwise the custom aggregate will be ignored.

Aggregate Library Project

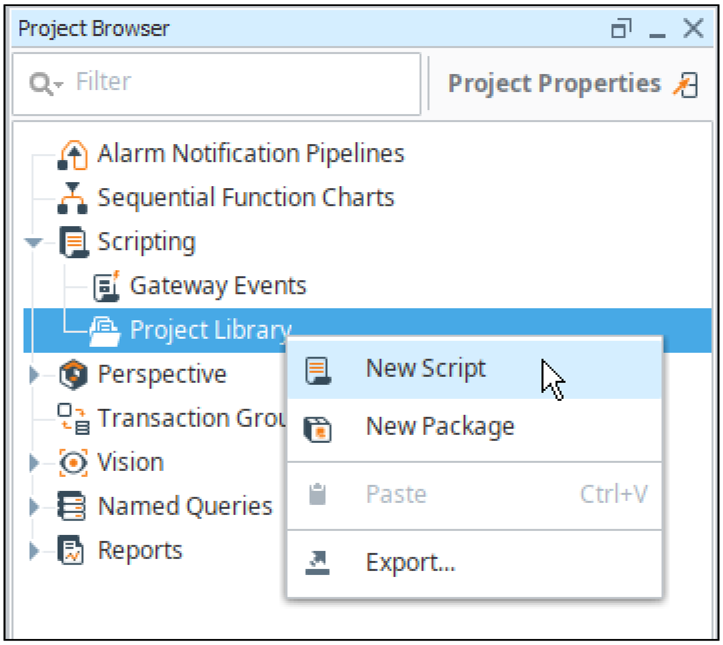
When using a Project Library, the library **must** reside in the Gateway's scripting project.

Library Name

The name of the project library must start with "`shared`", in lowercase. However there can be additional characters following the word "shared". For example: "shared_myLib"

Examples

1. Add a project script, by right clicking the **Project Library** item in the Project Browser, and choosing the **New Script** option.

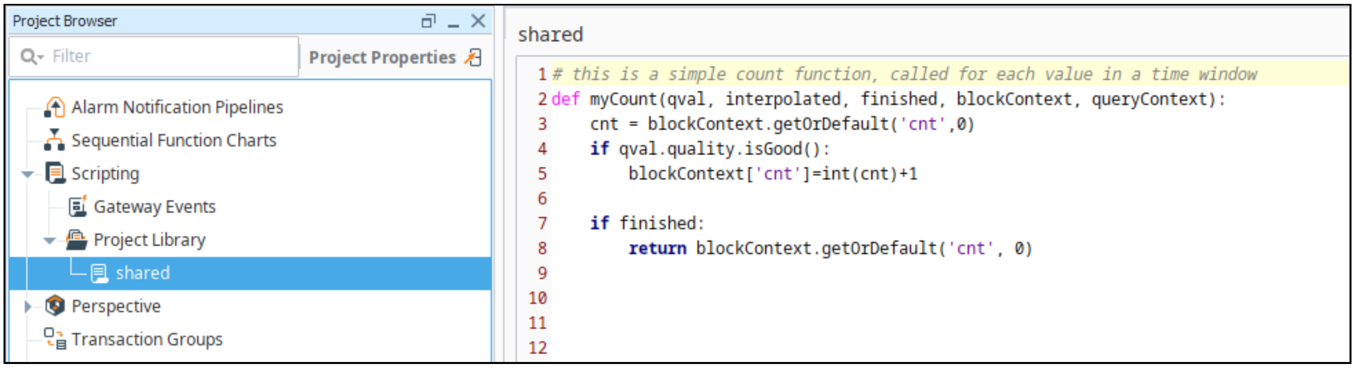


2. Enter a name for the script and click **Create Script** (we named ours *shared* in the example).
3. Enter the following code block:

Example

```
# this is a simple count function, called for each value in a time window
def myCount(qval, interpolated, finished, blockContext, queryContext):
        cnt = blockContext.getOrDefault('cnt',0)
        if qval.quality.isGood():
                blockContext['cnt']=int(cnt)+1

        if finished:
                return blockContext.getOrDefault('cnt', 0)
```



```
# this is a simple count function, called for each value in a time window
def myCount(qval, interpolated, finished, blockContext, queryContext):
    cnt = blockContext.getOrDefault('cnt',0)
    if qval.quality.isGood():
        blockContext['cnt']=int(cnt)+1

    if finished:
        return blockContext.getOrDefault('cnt', 0)
```

The custom function could be used by using the example below:

Example

```
#Return tag history using a custom aggregate function you wrote.

system.tag.queryTagHistory(paths=['MyTag'], rangeHours=1, aggregationModes=['shared.myCount'], returnSize =
100)
```

Creating an Aggregate Function on the Fly

Example

```
#Create a function on the fly to pass in as a custom aggregate.

wrapper = """\
python:def wrapper(qval, interpolated, finished, blockContext, queryContext):
        return shared.aggregates.customFunction(qval, interpolated, finished, blockContext, queryContext)
"""
system.tag.queryTagHistory(paths=['MyTag'], rangeHours=1, aggregationModes=[wrapper], returnSize = 100)
```

Tag History Providers

The Tag Historian module uses a concept of "History Providers" as a means to keep track of different storage configurations. Each provider is responsible for maintaining its own set of tables and records.

Tables used by the Tag Historian system are described on the Ignition Database Table Reference page.

On this page ...

Creating a History Provider

Most history provider types need to be manually created. New providers can be created on the Gateway:

1. Go to the **Config** section of the Gateway Webpage and select **Tags > History**
2. Click the "Create new Historical Tag Provider" link
3. Select the type, and click **Next**
4. Give the new provider a name, and make any other configuration changes
5. Click **Create New Historical Tag Provider** button when finished.

Datasource History Providers

Datasource History Providers can not be created or deleted manually. Instead they are tied to a database connection. They are automatically created when a connection to a database is configured on a gateway. Deleting a database connection from a gateway will also delete the Datasource History Provider, although any tables used by the provider will persist in the database.

Datasource History Providers Settings

The following table lists the settings for the Datasource History Providers. To access these settings, go to the **Config** tab of the Gateway Webpage and select **Tags > History**. Then click the **Edit** button for the provider you want to update.

Main	
Provider Name	Name of the Tag History Provider. By default, this will match up with the name of the database connection.
Enabled	If the check box is selected (enabled), the provider is turned on and accepts tag history data. If disabled, the database is not shown in the list of history providers when configuring tag history from the Designer. Also, any data logged to the provider, will error out and be quarantined by the store and forward engine, if possible.
Description	A user created description of the provider.
Data Partitioning	
Enable Partitioning	To improve query performance, Tag Historian can partition the data based on time. Partitions will only be queried if the query time range includes their data, thereby avoiding partitions that aren't applicable and reducing database processing. On the other hand, the system must execute a query per partition. It is therefore best to avoid both very large partitions, and partitions that are too small and fragment the data too much. When choosing a partition size, it is also useful to examine the most common time span of queries.
Partition Length and Units	The size of each partition, the default is one month. Many systems whose primary goal is to show only recent data might use smaller values, such as a week, or even a day.
Enabled Pre-processed Partitions	Enables pre-processed partitions.
Pre-processed Window Size (seconds)	When pre-processed partitions is turned on, this setting defines the window size.

Data Pruning	
Enable Data Pruning	Partitions with data older than a specific age are deleted. The check box is not selected/enabled by default. **Note:** Data pruning works by deleting old partitions. Therefore, data will only be removed when a partition has no data younger than the prune age.
Prune Age and Units	The maximum age of data. As mentioned, the data is deleted by the partition, and could therefore surpass this threshold by quite a bit before all of the data in the partition is old enough to be dropped.
Advanced	
Enable Stale Data Detection	If enabled, tracks Tag Group executions to determine the difference between unchanging values, and values that are flat due to the system not running.
Stale Detection Multiplier	The multiplier for Tag Group rate used to determine when values are stale. If Tag Group execution is not recorded within this amount of time, values will be considered bad on query.

Pre-Processed Partitions

Datasource history providers can opt into "pre-processed partitions". With pre-processed partitions enabled, records stored by the provider are stored in two different sets of partitions: the normal partitions, and a summarized "pre-processed" partition. In other words, the data that is stored in the normal partitions is summarized and then placed into an additional table in the database. While this takes up more space in the database, it can improve query speeds dramatically by reducing the amount of resulting data points. While enabled, tag history queries that request data in intervals greater than the specified **Pre-processed Window Size** setting will use the pre-processed tables.

For example, say a system.tag.queryTagHistory call was made with intervalSeconds set to 90. If the Pre-processed Window Size setting on the datasource history provider is set to something less than 90 seconds, such as the default value of 60 seconds, then the function would retrieve records from the pre-processed partitions. On the other hand, if the same function was used with intervalSeconds set to a value less than the Pre-processed Window Size, then the raw tables would be used. Both of these examples would retrieve data from the historian, but the first query's result set would be smaller than the second example's query.

When pre-processed partitions are enabled, a new partition table is created to store the pre-processed records. The pre-processed partitions use the following naming convention: `sqlth_data_driverID_YYYY_MM_windowSize`

More details on the pre-processed tables can be found on the Ignition Database Table Reference page.

Internal Historian Provider

The Internal Historian is a historian that stores data within the Ignition installation directory, via SQLite. It is available on standard Ignition Gateways.

Note: Records stored by the Internal Historian Provider don't contain information about Tag group execution, meaning stale execution data is not recorded.

The Internal Historian provider can choose to prune records when either the **Time Limited Enabled?** or **Point Limit Enabled?** settings are enabled. The check to prune records in these cases occur when the system tries to store a record.

Internal Historian Provider Settings

Main	
Provider Name	Name of the internal history provider.
Enabled	If the check box is selected (enabled), the provider is turned on and accepts tag history data. Default is true.

Description	A description of the provider.
Limits (Requires Historian Module Licensing)	
Time Limited Enabled?	Whether or not time limit is enabled. Default is true.
Time Limit Size	Size of the time limit. Unit (seconds, weeks, etc) is set in the Time Limit UnitsDefault is 1.
Time Limit Units?	Options are milliseconds, seconds, minutes, hours, days, weeks, months, or years. Default is WEEK.
Point Limit Enabled?	Whether or not point limit is enabled. Default is true.
Point Limit Size	Maximum number of data points the historian will store. Default is 10,000,000.
Sync Settings	
Remote Sync Enabled	Allows you to turn Tag History Synchronization on or off. Default is false.
Remote Gateway Name	The Gateway to target for remote synchronization. Must have the Tag Historian module installed, and allow remote storage. The Ignition Gateway's security settings will also need to be configured to allow remote storage.
Remote Provider Name	The remote history provider to sync data to.
Sync Frequency	The frequency with which data will be sent to the remote Gateway. This setting will be used in conjunction with the sync schedule, if enabled. Default is 10.
Sync Frequency Units	The unit of time that will be used with the Sync Frequency. Options are milliseconds, seconds, minutes, hours, days, weeks, months, or years. Default is SEC.
Max Batch Size	The maximum number of data points that will be sent per batch to the remote Gateway. Default is 10,000.
Enable Schedule	If enabled, the data will only be synchronized during the times specified by the pattern provided. Default is false.
Schedule Pattern	A comma separated list of time ranges. Examples: • 9:00-15:00 • 9pm-5am • 20.30-04.30

Remote History Provider

A Remote History Provider is a link to a historical provider on another Gateway. Since it is retrieving historical Tag data from another provider, its only configuration is to ensure it is pointed at the correct history provider on the remote system. You can't change any of the settings like partition length and prune age, but would instead have to change those settings on the original history provider on the remote Gateway. By default, the remote history provider will fall under the Default Security Zone and be read only.

Configuring a remote history provider involves some extra steps, when compared to the other provider types. Since they involve a tag provider on another Gateway, you'll be asked to select the target gateway. The list of Gateways is filled with gateways discoverable on the gateway network. If the Gateway is not currently available or displayed, you can specify its name manually.

Remote History Provider Settings

Main	
Provider Name	Name of the Tag History Provider.
Enabled	If the check box is selected (enabled), the provider is turned on and accepts Tag history data. Default is true. If disabled, the database is not shown in the list of history providers when configuring Tag history from the Designer. Also, any data logged to the provider, will error out and be quarantined by the store and forward engine, if possible.

Description	A description of the provider.
Remote Gateway	
Remote Gateway Name	The name of the remote Gateway.
Remote History Provider	The name of the provider on the remote Gateway. This does not have to match the provider name on the local Gateway.
Storage	
Allow Storage	If false, the provider will only be used for querying historical data. If true, the provider will create a store and forward pipeline for sending data to a remote Gateway. Default is true. This setting also requires that Service Security on the remote system allows for storage to the specified provider. When enabled, storage for this tag in the remote system will not include execution rate (which is normally logged into the sqth_sce table by local providers).
Max Bundle Size	The maximum number of data points that can be sent per request. This value is used in conjunction with the store and forward settings to dictate how much data is sent at once. Set to 0 for an unlimited size.

Tag History Splitter

Like the Remote History Provider, the Tag History Splitter Provider doesn't store history on its own, instead relying on having other providers already configured. A Splitter provider simply logs Tag History into multiple other History Providers.

When setting up a Tag to store history, selecting this provider will write the same data to both providers that it has selected. The Tag History Splitter is useful for automatically creating a backup of your data, or for reading history from two separate providers.

Tag History Splitter Provider Properties

Below are the properties available on the Historical Tag Provider.

Main	
Provider Name	Name of the connection.
Enabled	Enables and disables the connection.
Description	Description of the connection. The description appears on the Historical Tag Providers page of the Gateway.
Storage	
First Connection	Data is stored to both connections equally. However, all tag history queries (tag history bindings, system.tag.queryTagHistory calls, reporting tag historian queries, etc.) execute against the first connection, unless a limit is imposed using the settings below, or the first connection is unavailable.
Second Connection	The second connection to store Tag history.
Querying	
Limit First Connection Query	If enabled, only queries that are within the time frame specified below will be executed against the first connection. Queries that go further back will execute against the second connection.
Limit Length and Units	The unit and length of the time frame limitation mentioned above.

DB Table Historian Provider

This provider allows the Tag Historian module to retrieve data from tables that weren't created by the module, such as tables created with Transaction Groups or third party systems.

Read more about this provider type on the DB Table Historian Provider page.

Historian Simulator

The Historian Simulator allows the Tag Historian module to create simulated records that can be used for testing.

Read more about this provider type on the Historian Simulator page.

OPC-HDA Properties

Establishes a connection to an OPC-HDA Server to read history data that may be stored there from a third party. Ignition can not write to this type of history provider.

Note: This type of provider can only be created if the OPC COM module is installed and the gateway is running on a Windows operating system.

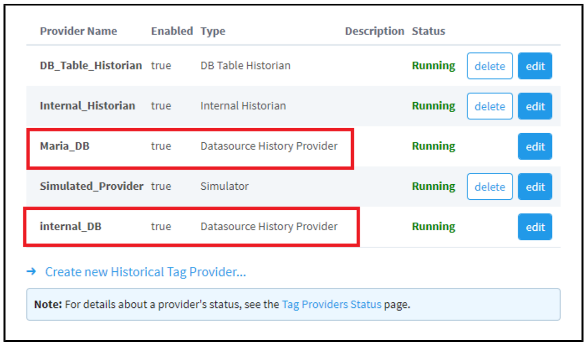
Main	
Provider Name	Name of the Tag History Provider.
Enabled	If the check box is selected (enabled), the provider is turned on and will expose historical data
Description	A description of the provider.
Server	
ProgId	A description of the provider.
Use Flat Browsing	Flat browsing returns all items at once. This is less efficient than normal browsing, but if a server only supports flat browsing, then this needs to be checked.
Remote Connection	
Remote Server	If selected, DCOM will be used to connect to the server on the specified Host with the given ProgId or CLSID.
Host Machine	The name or IP address of the machine hosting the server. Leave empty for local machine.
CLSID	The CLSID of the server. If not specified, will be obtained using the ProgId.

DB Table Historian Provider

The following feature is new in Ignition version **8.1.0**
Click here to check out the other new features

The DB Table Historian Provider acts as a bridge between tables in a database connection and the Tag Historian Module, mapping columns in the tables as "available Tags", thus allowing Tag History Queries to access the content of the table. Usually Tag History Queries can interact only with tables created by the Tag Historian. Thus, a table created with a Transaction Group, or by some other means (e.g., manually creating the database tables, tables generated by third party systems, etc) couldn't be accessed via things like a Tag History Binding. The DB Table Historian Provider solves this problem.

Utilizing this provider requires a **Datasource History Provider**, which is the type of historical provider that is automatically created whenever a Database connection is configured. You can check the type of any History Provider under the **Config** section of the Gateway, on the **Tags > History** page.



Provider Name	Enabled	Type	Description	Status	
DB_Table_Historian	true	DB Table Historian		Running	delete edit
Internal_Historian	true	Internal Historian		Running	delete edit
Maria_DB	true	Datasource History Provider		Running	edit
Simulated_Provider	true	Simulator		Running	delete edit
internal_DB	true	Datasource History Provider		Running	edit

→ Create new Historical Tag Provider...

Note: For details about a provider's status, see the Tag Providers Status page.

On this page ...

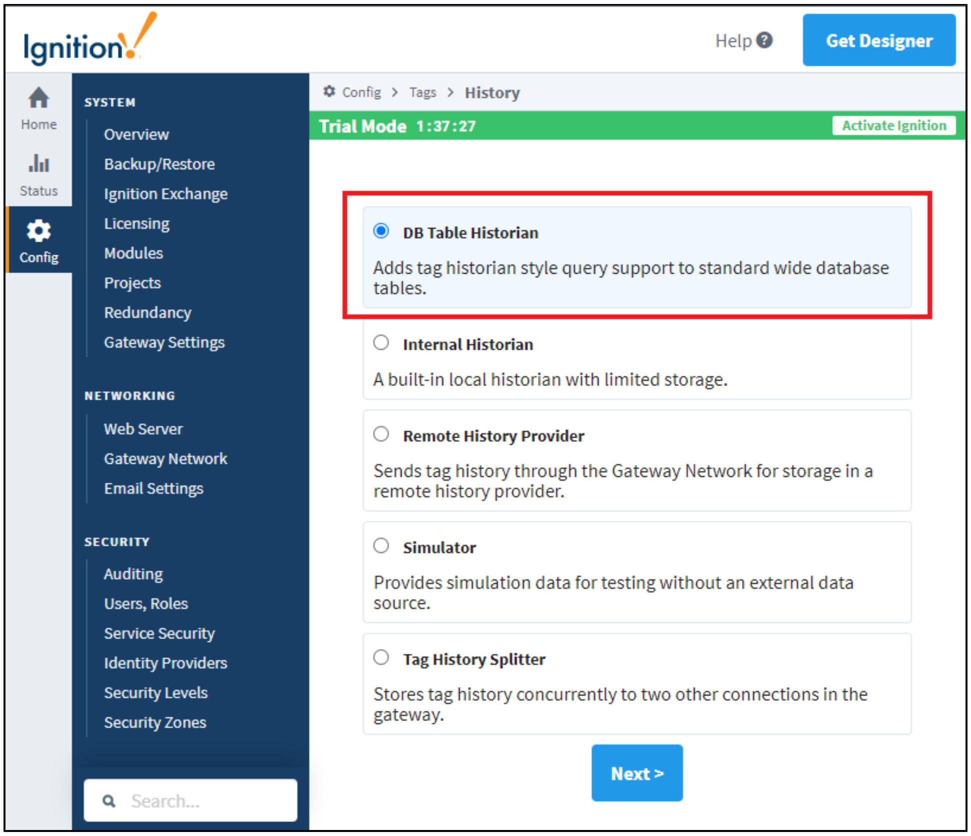
INDUCTIVE UNIVERSITY

DB Table Historian

Watch the Video

Configuring a DB Table Historian Provider

1. On the Gateway Webpage, navigate to the Config section.
2. Under the TAGS heading in the sidebar, click on **History**.
3. On the History Provider's listing, click **Create new Historical Tag Provider**.
4. From the listing of available types, select **DB Table Historian**, and click **Next**.



● **DB Table Historian**

Adds tag historian style query support to standard wide database tables.

○ **Internal Historian**

A built-in local historian with limited storage.

○ **Remote History Provider**

Sends tag history through the Gateway Network for storage in a remote history provider.

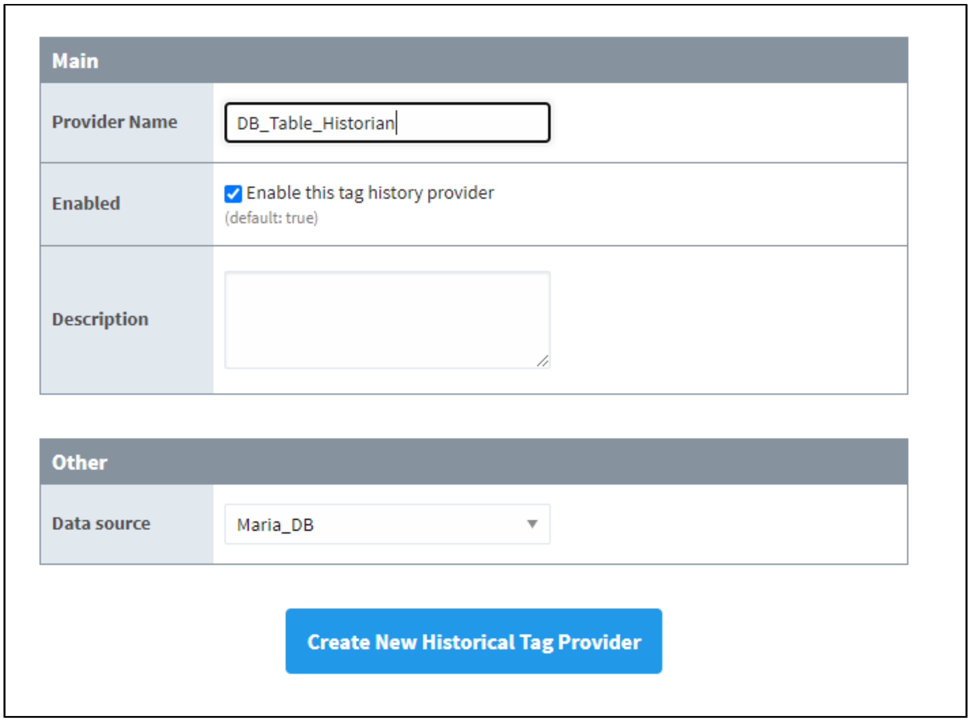
○ **Simulator**

Provides simulation data for testing without an external data source.

○ **Tag History Splitter**

Stores tag history concurrently to two other connections in the gateway.

5. Enter a unique name for the provider in the Provider Name field.
6. In the Data source field, select the backing data source. You would select which ever Datasource History Provider contains the tables you want exposed to the historian system. Once ready, click the **Create New Historical Tag Provider** button, which completes the configuration process.



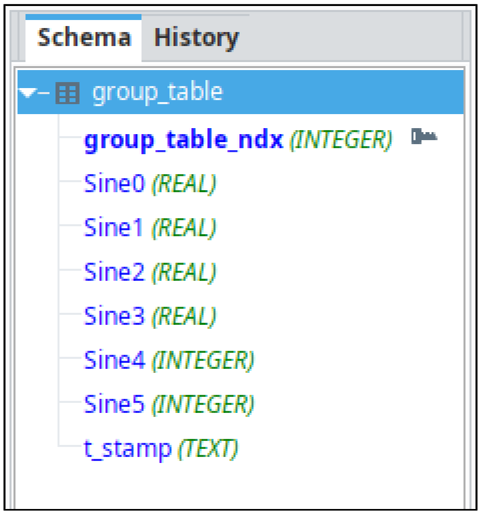
Main	
Provider Name	DB_Table_Historian
Enabled	☑ Enable this tag history provider (default: true)
Description	

Other	
Data source	Maria_DB

Retrieving Records

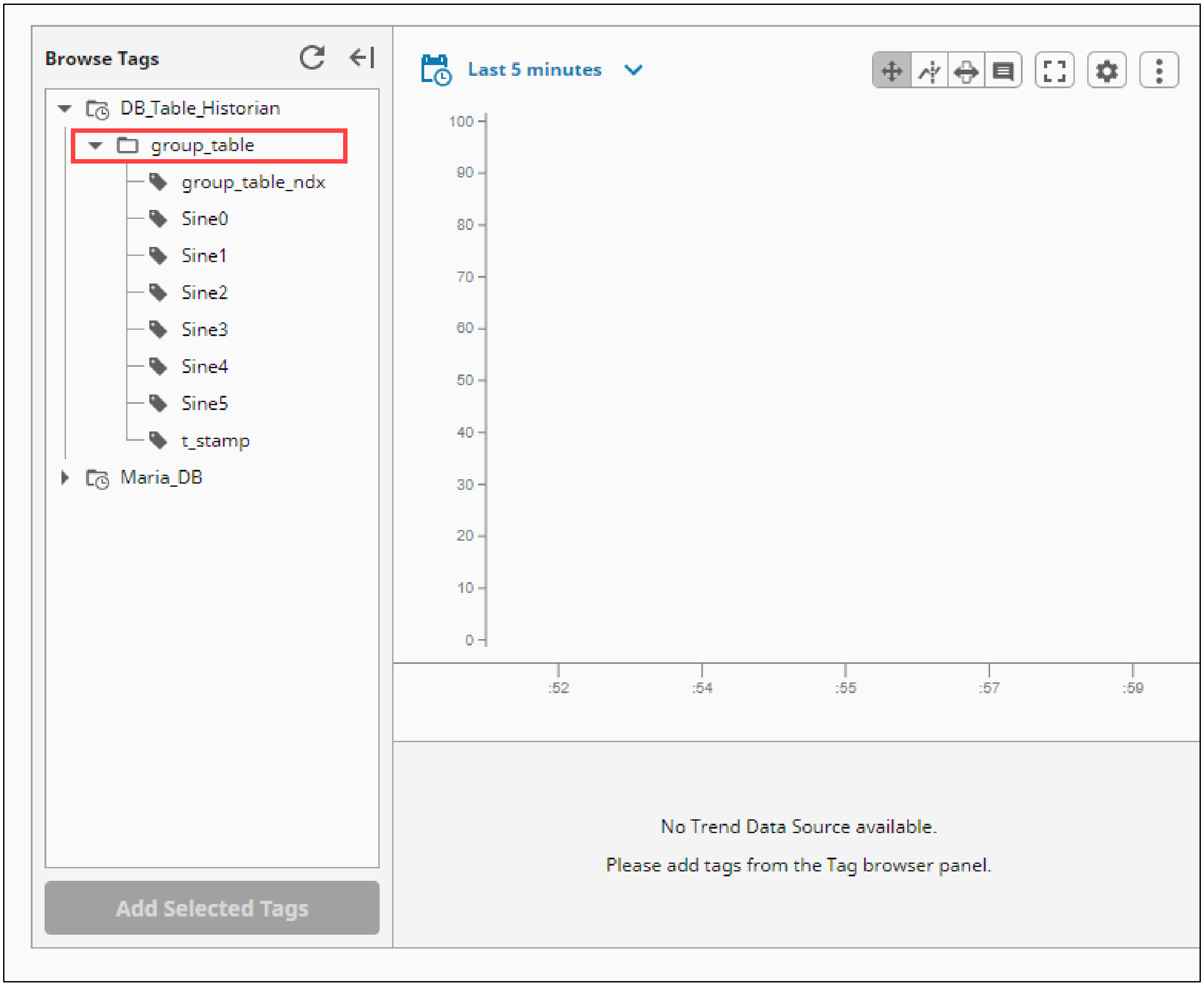
Once configured, the DB Table Historian will expose any tables found in the associated data source (specifically, the **Data Source** listed on the configuration page of the DB Table Historian). From this point on, any tables found in the data source will be available for browsing via the various Tag History interfaces found throughout Ignition.

For example, we could store some records in a database connection with a Transaction Group, which would create a table like the following:



group_table
- **group_table_ndx** *(INTEGER)*
- Sine0 *(REAL)*
- Sine1 *(REAL)*
- Sine2 *(REAL)*
- Sine3 *(REAL)*
- Sine4 *(INTEGER)*
- Sine5 *(INTEGER)*
- t_stamp *(TEXT)*

Then, we could use the Perspective Power Chart built-in Tag browser panel to detect the table, which exposes the columns as "tags":



- DB_Table_Historian
  - group_table
    - group_table_ndx
    - Sine0
    - Sine1
    - Sine2
    - Sine3
    - Sine4
    - Sine5
    - t_stamp
- Maria_DB

No Trend Data Source available.

Please add tags from the Tag browser panel.

Path Components

The DB Table Historian Provider attempts to map each column in a database table to a Tag. When querying the results, the historical Tag path used is composed of multiple components, which each represent some identification of the data source. The DB Table Historian is molded after the following:

```
histprov:[historyProvider]:/table:[tableName]:/column:[columnName]:/timestamp:[timestampColumnName]:
/keycolumn:[keyColumnName]:/keyvalue:[keyColumnValue]
```

Each of the components are described below.

Component	Description
histprov	The name of the history provider that should be queried.
table	The name of the database table in the the history provider.
column	The name of the column on the table, in the history provider.
timestamp	A column on the table that will be used as the source of the timestamp for the query. By default, the query will look for a column named **t_stamp** to use for the timestamp component. It's highly recommended to include this component if the table doesn't contain a timestamp column named "t_stamp", otherwise the query will fail.
keycolumn	An optional component that allows you to specify a single column on the table to use for simple filtering. Used in conjunction with keyvalue. There can only ever be a single keycolumn for any given path. More complex filtering can be accomplished by instead using a Named Query.
keyvalue	An optional component that works with keycolumn, allowing the query to only return rows if the keycolumn contains the value specified on this component. Value must be an integer.

As far as historical pathing goes, Tags created by this provider may look something like below, with dedicated **table** and **column** components in the path.

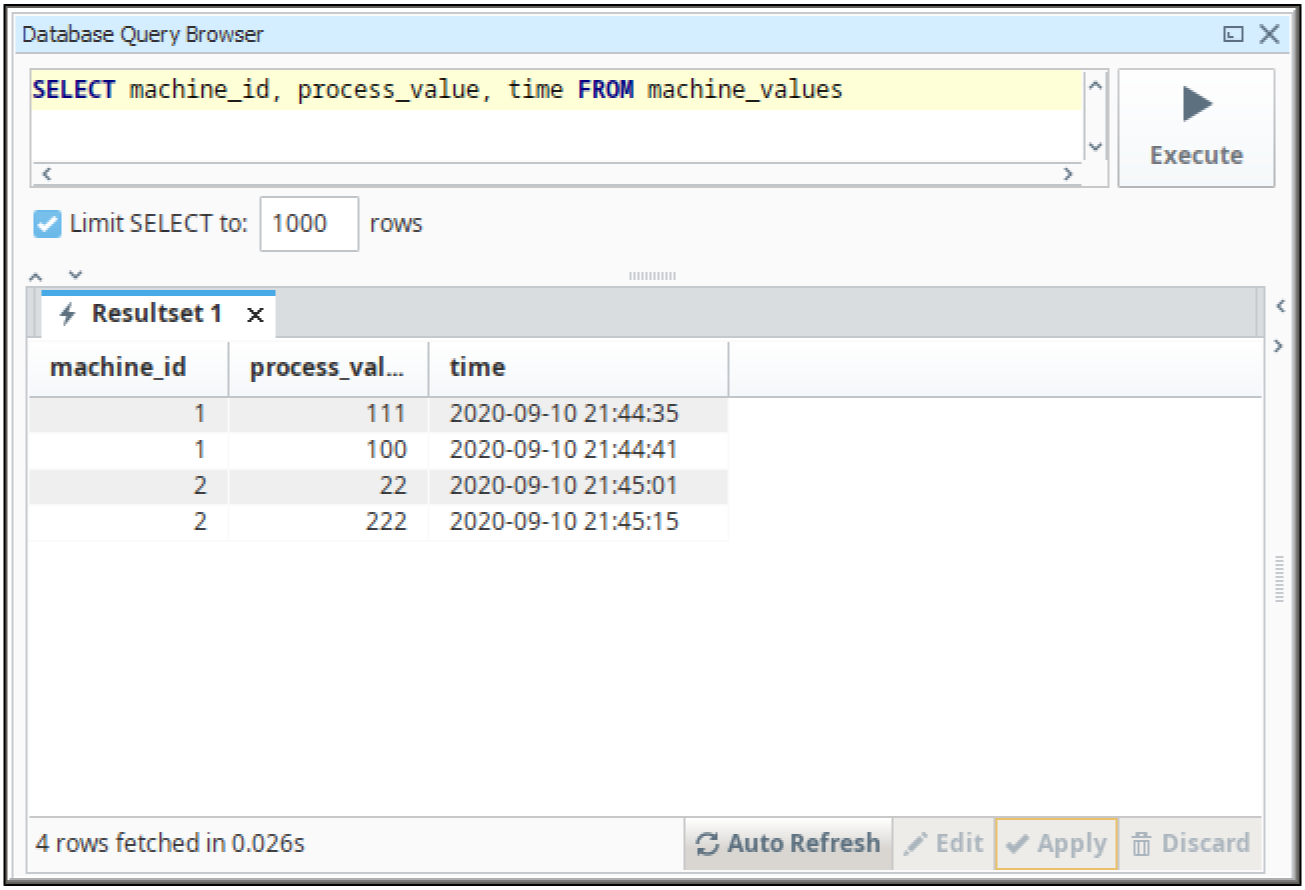
```
histprov:DB_Table_Historian:/table:group_table:/column:Sine1:/timestamp:my_timestamp_column
```

Example

Say we have a database table like the following.

```
SELECT machine_id, process_value, time FROM machine_values
```

machine_id	process_value	time
1	111	2020-09-10 21:44:35
1	100	2020-09-10 21:44:41
2	22	2020-09-10 21:45:01
2	222	2020-09-10 21:45:15



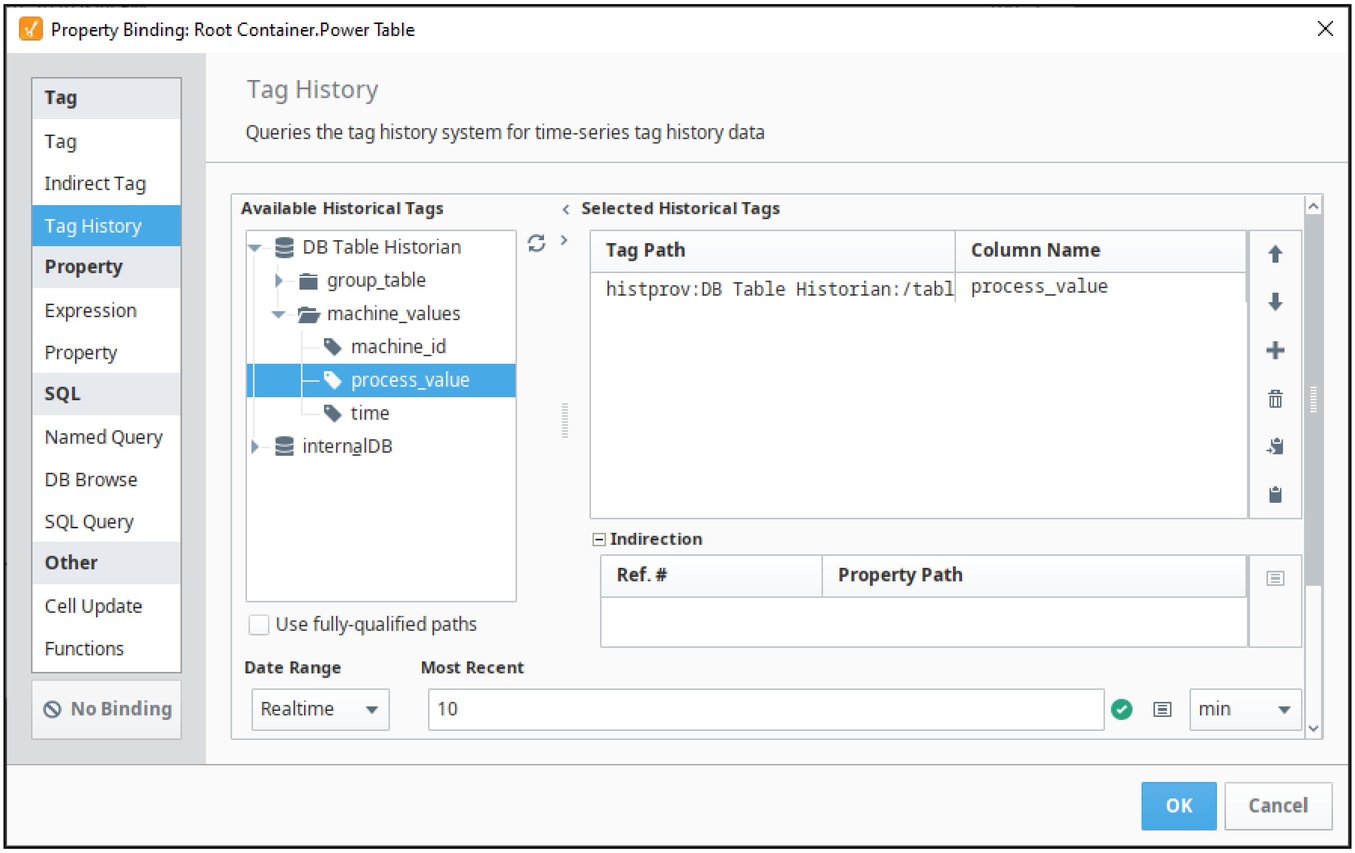
```
SELECT machine_id, process_value, time FROM machine_values
```

☑ Limit SELECT to: 1000 rows

machine_id	process_val...	time
1	111	2020-09-10 21:44:35
1	100	2020-09-10 21:44:41
2	22	2020-09-10 21:45:01
2	222	2020-09-10 21:45:15

4 rows fetched in 0.026s

We could use the DB Table Historian provider to expose our machine_values table to a Vision Tag History Binding.



Tag History

Queries the tag history system for time-series tag history data

Selected Historical Tags

Tag Path	Column Name
histprov:DB Table Historian:/tabl	process_value

Indirection

Ref. #	Property Path

☐ Use fully-qualified paths

Date Range: Realtime

Most Recent: 10 min

The resulting historical Tag path for process_value (after dragging it over to the **Selected Historical Tags** table) would look like the following:

```
histprov:DB Table Historian:/table:machine_values:/column:process_value
```

We would need to explicitly state that the "time" column should be used by the binding, so we could double click on the cell under the "Tag Path" header, and change the Tag Path to the following, which would allow the query to feature the records accurately.

```
histprov:DB Table Historian:/table:machine_values:/column:process_value:/timestamp:time
```

We could further modify this Tag path, so that only entries with a "machine_id" of 1 are returned, but changing the path to the following:

```
histprov:DB Table_Historian:/table:machine_values:/column:process_value:/timestamp:time:/keycolumn:
machine_id:/keyvalue:1
```

Historian Simulator

The following feature is new in Ignition version **8.1.0**
Click here to check out the other new features

On this page ...

Historian Simulator

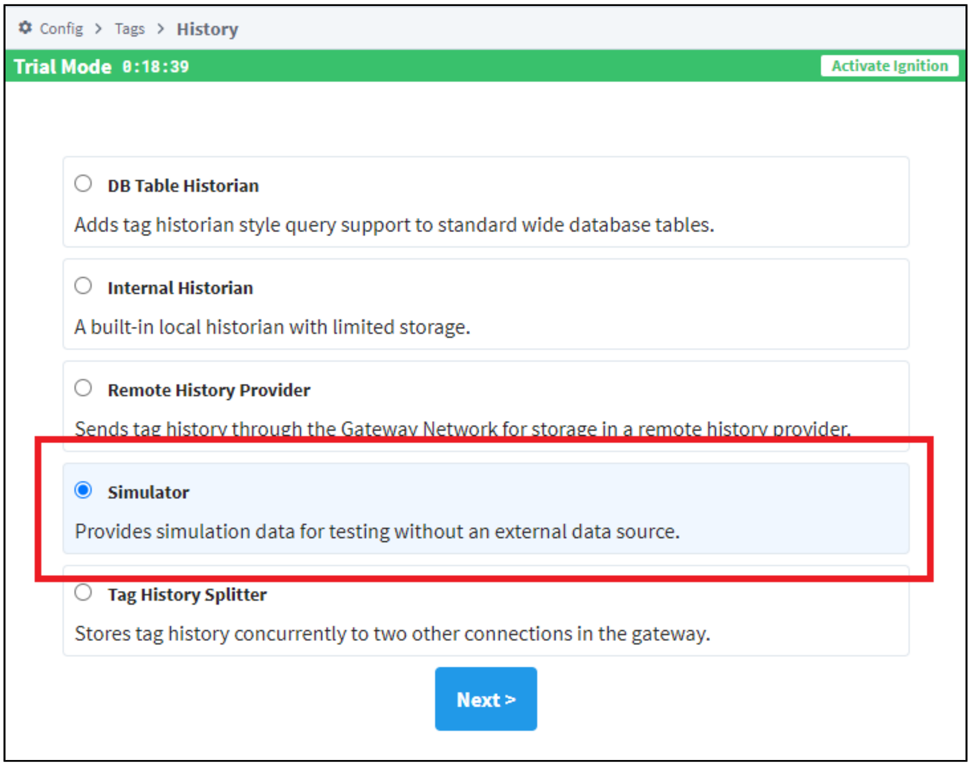
Watch the Video

The Historian Simulator is a provider that will generate simulated historical records, without the need for an external database or data source. The Simulator doesn't actually store any records. Rather, subsystems in Ignition can make requests to the simulator, and it will generate a result set based on the Tag path(s) provided.

The simulator is great for testing, as the data generated is reliably aligned to the specified period. Each generation of data for a given set of parameters and given time range will be repeatable. In addition, the "resolution" of the raw data, or the frequency of the generated datapoints, is configurable allowing for as dense or sparse raw values.

Configuring a Historian Simulator Provider

1. On the Gateway Webpage, navigate to the Config section.
2. Under the **TAGS** heading in the sidebar, click on **History.**
3. On the History Provider listing, click **Create new Historical Tag Provider.**
4. From the listing of available types, select **Simulator**, and click **Next.**



Config > Tags > History

Trial Mode 0:18:39 Activate Ignition

○ **DB Table Historian**
Adds tag historian style query support to standard wide database tables.

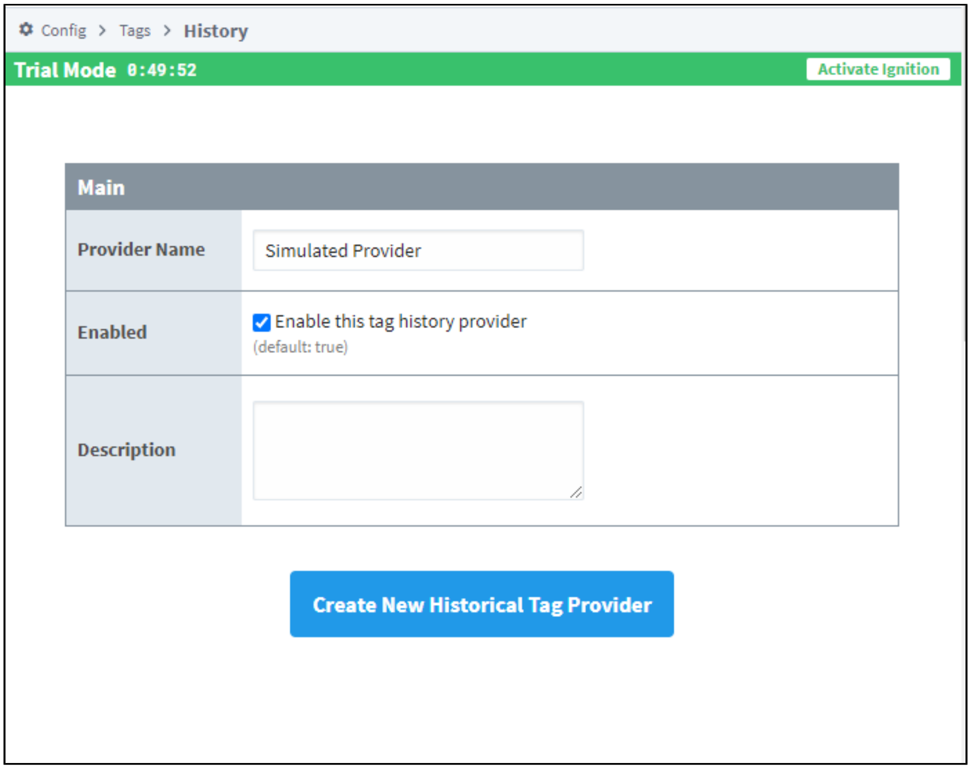
○ **Internal Historian**
A built-in local historian with limited storage.

○ **Remote History Provider**
Sends tag history through the Gateway Network for storage in a remote history provider.

● **Simulator**
Provides simulation data for testing without an external data source.

○ **Tag History Splitter**
Stores tag history concurrently to two other connections in the gateway.

5. The simulator requires a unique name from other historical providers on the same Gateway, but otherwise does not require any other configuration settings. The bulk of configuration occurs when attempting to request records from the simulator. For now just provide a name, and click the **Create New Historical Tag Provider** button to finish configuring the provider.



Main

Provider Name	Simulated Provider
Enabled	☑ Enable this tag history provider (default: true)
Description	

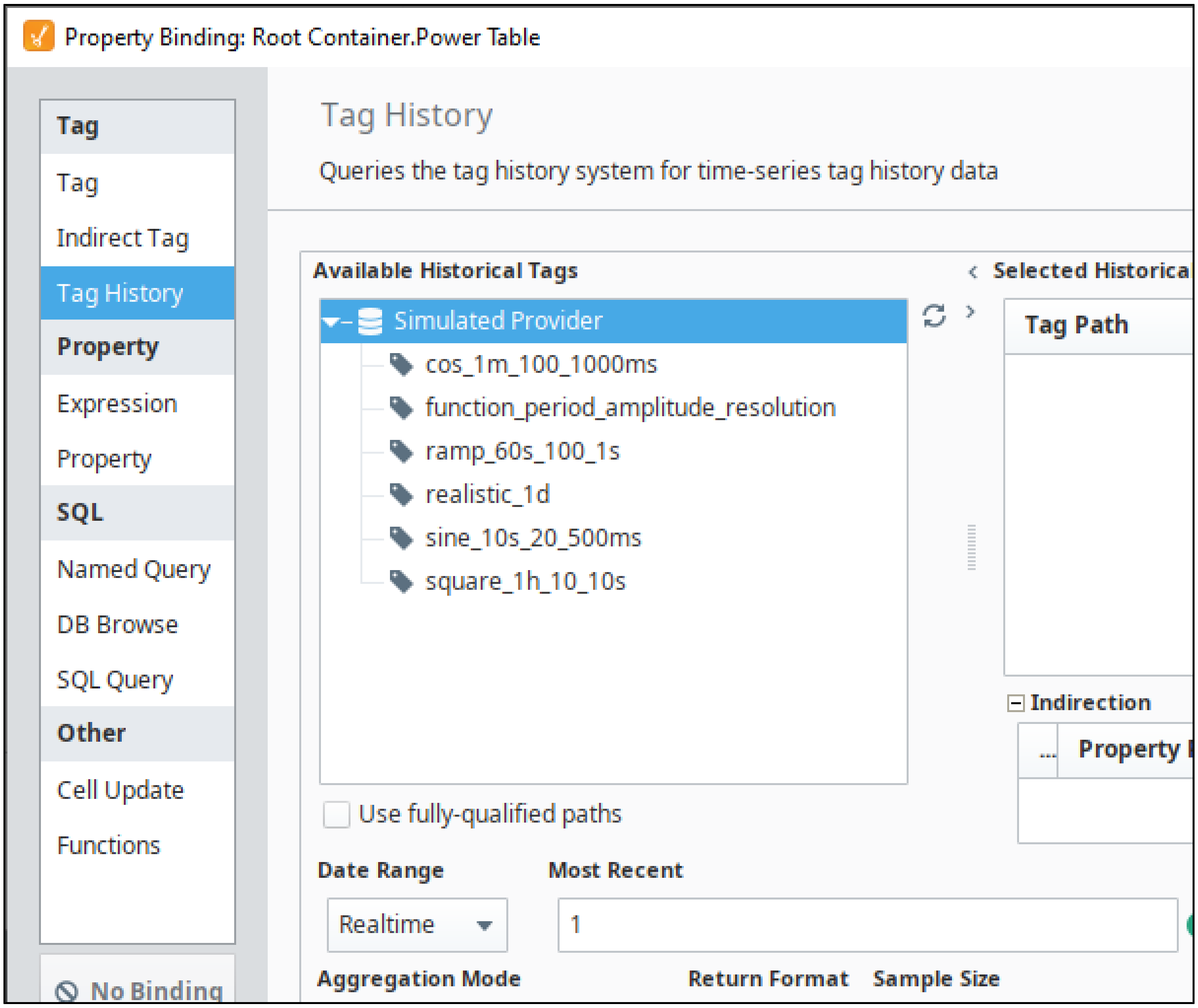
6. You'll be redirected back to the History Provider listing. You're now ready to use the simulator.

Retrieving Simulated Data

Once a Simulator Provider has been configured, you need a Tag Historian Query to start retrieving records from it. Lots of different interfaces can request Tag Historian Queries, but for the sake of simplicity we'll only focus on Tag history bindings in Vision. Browsing for available historical Tags using a Vision Tag History binding reveals the following Tags:

Simulated Provider

- cos_1m_100_1000ms
- function_period_amplitude_resolution
- ramp_60s_100_1s
- realistic_1d
- sine_10s_20_500ms
- square_1h_10_10s



Property Binding: Root Container.Power Table

Tag History

Queries the tag history system for time-series tag history data

Date Range: Realtime

Most Recent: 1

To get started quickly, you can use the default Tags in the list above. The one exception would be the "`function_period_amplitude_resolution`" Tag, as that's simply there for syntax reference. The rest of the Tags utilizes the simulator's pathing syntax.

Simulator Path Syntax

The built-in "Tags" provided by the simulator are a great example of the simulator can do, but it is worth knowing that the data returned by the simulator can be customized. Doing so involves modifying the Tag path in the Tag history query.

As a refresher, history providers require a path that contains multiple components. Together, the components identify where the data came from. For example, a historical path could have a `histprov` component that represents the history provider associated with the data, and a `Tag` component that represents the Tag path of the Tag. A hypothetical historical path would look like the following:

```
histprov:my_provider:/tag:my_tag_path
```

With that understanding, retrieving records from the simulator involves providing the name of the simulator configuration on the gateway (the `histprov`) and a Tag path. Again, the simulator doesn't store any records. Rather, it looks at the Tag path provided, parses it, and returns a result set based off of contents in the path.

The simulator expects Tag paths to use the following notation, with each "parameter" delimited by an underscore:

```
function_periodTime_amplitude_resolutionTime
```

Each parameter is described below:

parameter	description
function	The function to use when generating the raw data. The following functions are available:

	- Ramp - Start at zero, and increment up to Amplitude over a the periodTime. Once Amplitude has been reached, start over at zero and repeat. - Sine - Generates a Sine wave over the period of time, with Amplitude representing the peak deviations from zero. - Cos - Generate a Cosine wave over a period of time, with Amplitude representing the peak deviations from zero. - Square - Half the time is spend at zero, the other half is spent at amplitude, across periodTime. - Realistic - Generates 400 random but somewhat realistic data points that repeat each period. Note that this function ignores both the amplitude and resolutionTime parameters.
periodTime	How often the data sequence starts over. Values for periodTime need a time unit.
amplitude	Used as a peak value by most of the functions.
resolutionTime	How often raw data points are generated. Like periodTime, this parameter requires a time unit.

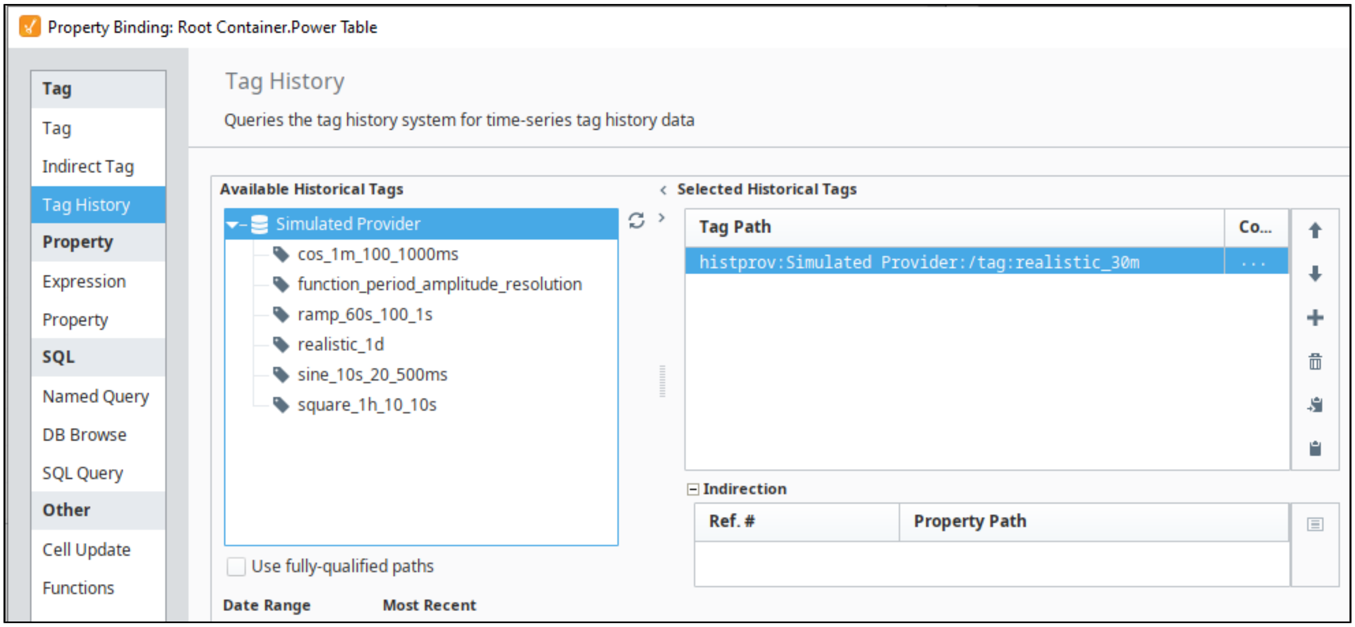
For those functions utilizing a time unit represented by certain notation. A description of each notation can be found below:

Notation	Time Unit
ms	Milliseconds
s	Seconds
m	Minutes
h	Hours
d	Days

Editing the Paths

With the knowledge of how the paths work, you can easily use the existing Tag history interfaces to request datasets with custom paths, usually by simply typing in the desired parameters.

For example, the image below shows Vision's Tag History Binding interface, but we modified the Selected Historical Tag path to use the realistic function over a period of 30 minutes, instead of the default one day.



Property Binding: Root Container.Power Table

Tag History

Queries the tag history system for time-series tag history data

Available Historical Tags

- Simulated Provider
  - cos_1m_100_1000ms
  - function_period_amplitude_resolution
  - ramp_60s_100_1s
  - realistic_1d
  - sine_10s_20_500ms
  - square_1h_10_10s

Selected Historical Tags

Tag Path	Co...
histprov:Simulated Provider:/tag:realistic_30m	...

Indirection

Ref. #	Property Path

☐ Use fully-qualified paths

Date Range Most Recent

Function Examples

For example, the Tag in the path below utilizes a sine function, over a period of 10 seconds, with an amplitude of 20, and a raw point will be generated every 500 milliseconds.

```
tag:sine_10s_20_500ms
```

The following would generate a square wave, with an amplitude of 10, repeating every day, while generating a raw value for every 5 minutes.

```
tag:square_1d_10_5m
```

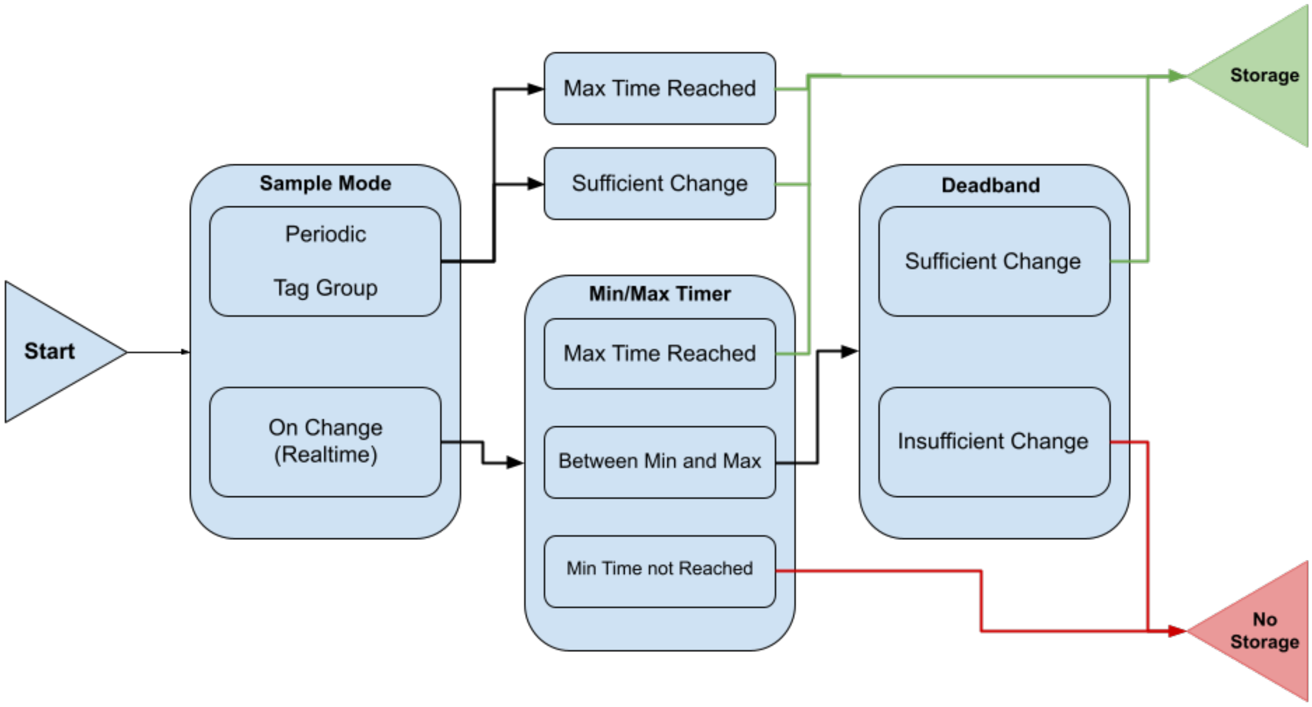
The realistic function is unique in that it only utilizes the periodTime parameter, therefore:

```
tag:realistic_30m
```

How the Tag Historian System Works

This page provides an overview on how the Tag Historian system evaluates tag changes, stores them, and later retrieves those values.

Deciding When to Store



The diagram above demonstrates the evaluation cycle for determining if the system needs to collect a sample starting with the sample mode type and working through timer settings and deadband change sufficiency.

When configuring history on a tag, these are some of the several checks that occur. These checks ultimately determine if the system should collect a sample. Each check is handled by something on the backend of Ignition called an "Actor". The notable actors here are **Sample Mode**, **Min/Max Timer**, and **Deadband**, which each correlate to similarly named properties on the tag's configuration.

Sample Mode

First the system needs to detect a value change on the tag, which is handled by the **Sample Mode** actor.

- If the tag is using a Periodic or Tag Group mode, then the storage evaluation occurs at the same rate specified by the mode's rate.
- If the tag is using an On Change mode, then storage evaluation occurs every time the tag changes value.
- In addition, there is a periodic timer for On Change modes running that checks if an On Change tag has remained static for a period of time longer than the configured maximum time between samples. If this occurs then the system immediately collects a sample, regardless of the other actors. The periodic timer evaluates all tags that share a tag provider, history provider, and rate simultaneously. So two tags that share a tag provider and history provider, but have different sample rates, will be evaluated separately.

Min/Max Timer

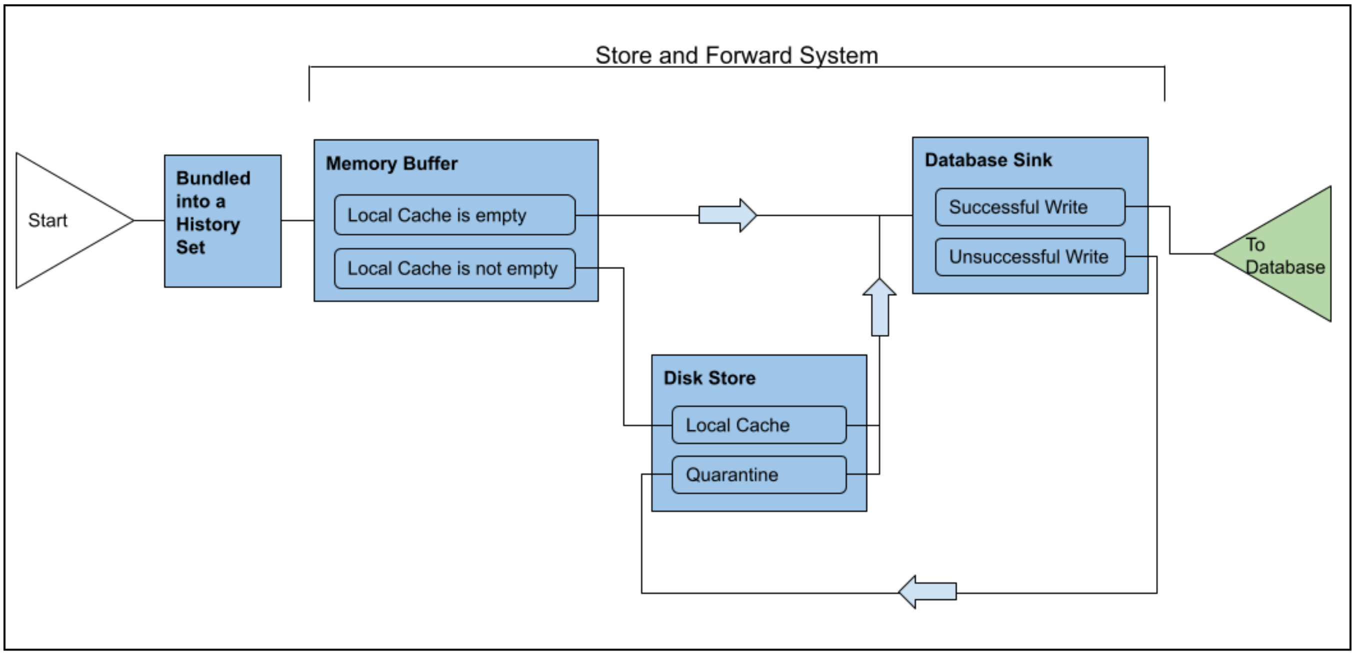
The next actor is the **Min/Max Timer**. Where applicable, if the mode's minimum time between samples has not yet been reached, then the new value is discarded. Otherwise, if the maximum time has been exceeded, then the system will collect a sample. The Max Time between samples will bypass the deadband, but not the sample rate. If the value is in between any minimum and maximum allowable sample periods, then the value is handed off to the next actor.

When using a Tag Group sample mode, there are two locations where a Max Time can be defined: On the tag's history settings, and on the Tag Group's history settings. The Tag Group's settings override the settings on the Tag, *except* when the Tag Group is using it's default values: in other words, neither the Max Time Between Samples or Max Time Units settings on the Tag Group have been changed.

Deadband

Next the **Deadband** actor examines the newly changed value in comparison to the previous value. The deadband will ultimately determine if the new value is sufficiently different enough than the previous value. If so, then a sample is collected.

Storing a Value



Once a sample is collected, it's added to a **History Set**, which is a collection of samples to store from a particular Tag Group/Scan Class (values using a "periodic" sample mode are all bundled together under an "_exempt_" group).

After being bundled, the History Set is processed by the Store and Forward System.

Memory Buffer

The History Set first reaches the Memory Buffer. If the Local Cache (in the Disk Store) is empty, then we know there are no other sets to process, and our new set is sent off to a Database Sink.

If the Local Cache has records, then the memory buffer will direct the newly acquired History Set to the Disk Store.

Disk Store

The Disk Store stores records on disk, in one of two areas.

The **Local Cache** is used to store sets that need to be sent to the Database Sink. Sets stay here until the Store and Forward system's Write Time or Write Size are met.

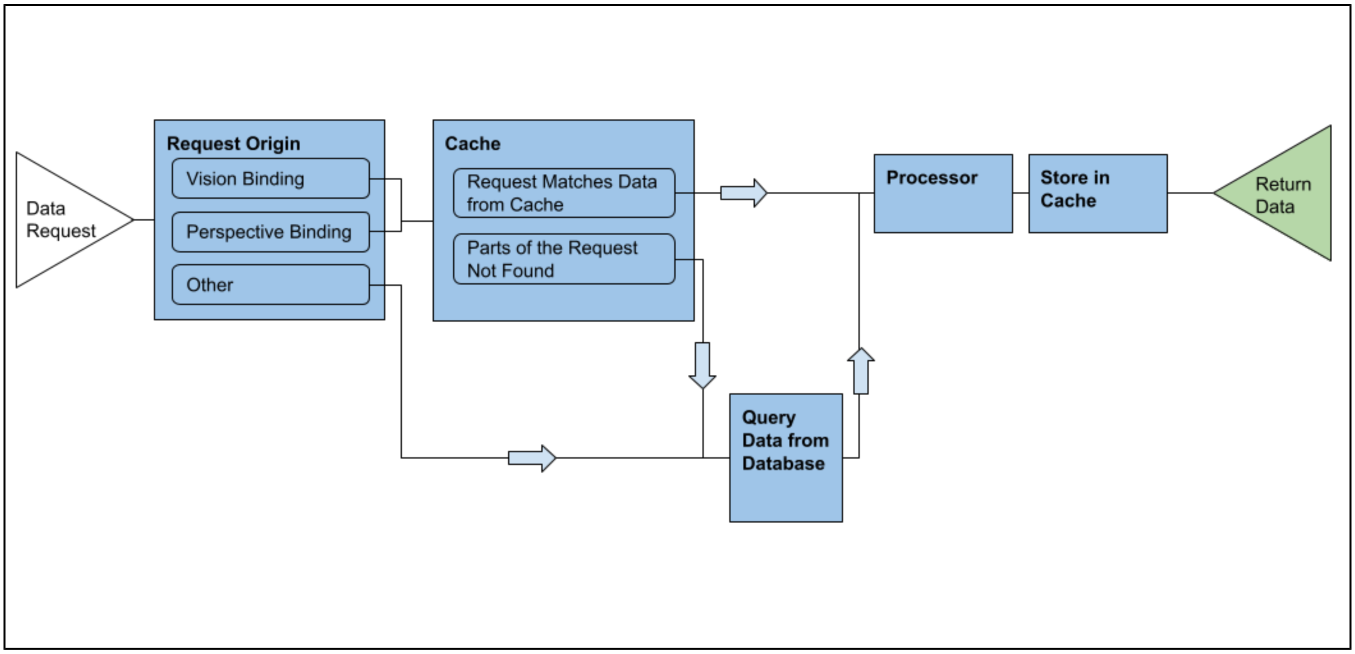
The **Quarantine** stores records that failed to be sent to the Database, typically due to an error reported by the database.

Database Sink

A Database Sink is responsible for several things:

- Bundle samples from a History Set into a SQL transaction
- Execute queries against the database.
- Keep tag id values in memory, so the sink doesn't have to constantly query the database for identifying features when a new record from an existing tag comes in.
- Samples that failed to be inserted into the database will be retried as individual queries before being moved to the Quarantine.

Retrieving a Value



Many features in Ignition can request data from the historian system. Requests must include some key attributes:

- Start datetime.
- End datetime.
- Tag paths.
- Sample size, determining how many datapoints should be returned.
- Aggregation mode, used in cases where many points are shown in a relatively small time frame.
- Return format, either wide or tall.

Request Origin

Tag History requests from Perspective and Vision Component Bindings can be cached for future use. Because of this the history system takes note of where the request originated from. If it was from a Tag History component binding then the system will check the appropriate cache. Other origins, such as the system.tag.queryTagHistory function don't have a cache, and instead immediately query from the database.

Of note, component bindings that opt out of using the cache (enabling the Vision "Bypass Tag History Cache" setting, or by disabling the Perspective "Cache & Share" setting) will skip the cache and query their results from the database.

Cache

Requests originating from a component binding are compared to existing caches. If data within the cache matches the request exactly, then the request is sent to the Processor.

If the cache is missing some of the requested data points, then they're retrieved from the database.

Query Data from Database

Here the system will write and execute a query to retrieve records from the database, based upon the start datetime and end datetime. The Tag Historian normally partitions data across multiple tables, so this step typically involves several SELECT queries to discover which partitions need to be consulted.

If pre-processed partitions are enabled, the system will determine if it should use those processed tables at this step.

Processor

The Processor has several responsibilities. Ultimately it needs to create a result set to return to the originator, but it also needs to process the data in the following ways:

- If some of the records from the query were retrieved from the cache, then the Processor is responsible for combining the cached records with the newly queried records.
- The Processor is responsible for placing records into time slices, as well as aggregating or interpolating the data as necessary. For example, if a query is requested to return 10 data points over a 10 minute time period, the processor would create 10 time slices that span one minute each. If there were multiple values collected within any of those slices, the values are aggregated using the selected aggregation method. If there aren't any records within that slice, interpolation will create a value.

Store in Cache

If the request originated from a system that has a cache, the result set created by the Processor is cached at this step. Regardless, after this step the result set is handed back to the object that originally made the request.

Storage Format

A listing of tables used by the Tag Historian can be found on the Ignition Database Table Reference page.

SQL Bridge (Transaction Groups)

Transaction Groups are the heart of the SQL Bridge module. They are execution engines that perform database tasks such as storing data historically, synchronizing database values to a device, and loading recipe values. A variety of group types, items types, and options means that Transaction Groups can be configured to accomplish a variety of tasks against a database table.

On this page ...

Transaction Group Overview

At their core, Transaction Groups are user-configured collections of data references (called **items**) that are then linked to columns in a table in a SQL database. Items can be OPC references, tag references, expressions, or SQL queries, and transaction groups allow precise control of data flow to, or from, specified items to a configured table. To set up and use Transaction Groups, SQL knowledge is not required; all a group needs to run is a database connection. Ignition can automatically create and manage the database table for each group.

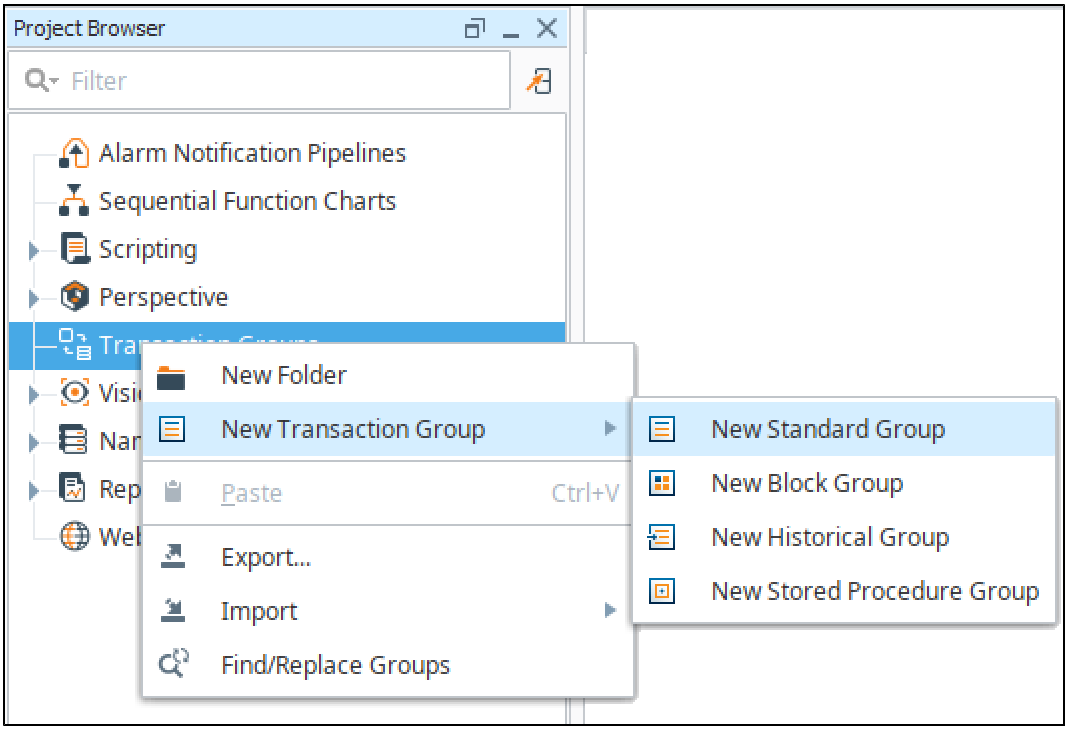
Transaction Groups are configured in the Ignition Designer. The group will then execute at a specific interval of time, or on a user-defined schedule. A trigger can be used to prevent planned executions when specified conditions are not met. On execution, the simplest groups (Historical groups) will create a new row in the database table with a separate column for each Item in the group. However, more complex variations are possible.

Transaction Group Applications

There are four types of Transaction Groups, and they each handle data a little differently.

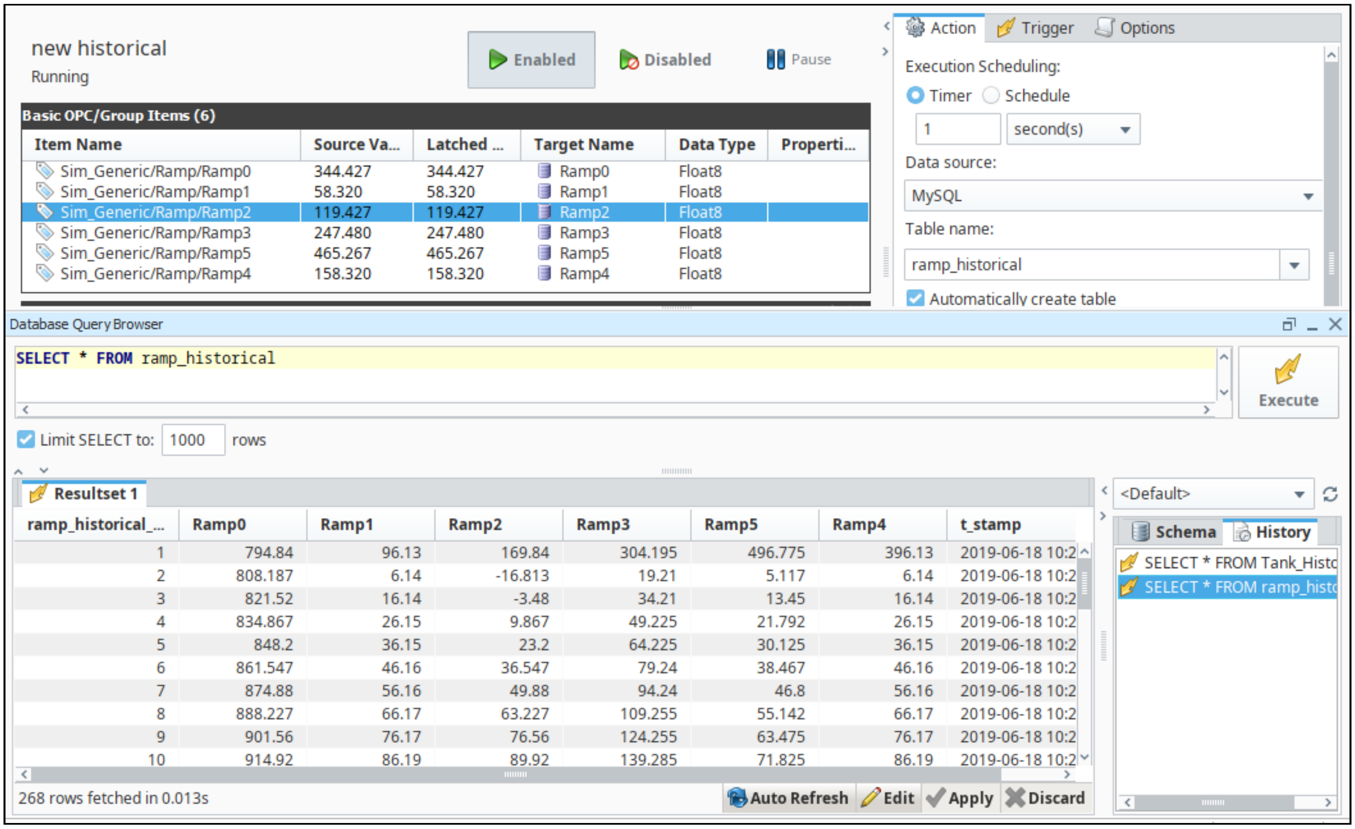
- **Historical Group** - A basic, one-way group that collects a data point from each item, and logs it to a column in the database.
- **Standard Group** - A flexible, two-way group that can log data to the database, or pull data from the database to write to OPC items or tags.
- **Block Group** - A standard group with an additional feature: each execution of the group can create, modify, or extract multiple rows from the database table, rather than just one.
- **Stored Procedure Group** - A group capable of invoking a stored procedure in the database, returning the results of any OUT or INOUT parameters to tags or OPC items.

Learn more about each type of group on the Types of Groups page.



Historical Data Logging

Transaction groups can function as a flexible alternative to the Tag Historian, which is more focused on speed and efficiency than simplicity. The Historical Group is explicitly designed for this purpose, but the Block Group can allow multiple rows to be logged at once.



Basic OPC/Group Items (6)

Item Name	Source Va...	Latched ...	Target Name	Data Type	Properti...
Sim_Generic/Ramp/Ramp0	344.427	344.427	Ramp0	Float8	
Sim_Generic/Ramp/Ramp1	58.320	58.320	Ramp1	Float8	
Sim_Generic/Ramp/Ramp2	119.427	119.427	Ramp2	Float8	
Sim_Generic/Ramp/Ramp3	247.480	247.480	Ramp3	Float8	
Sim_Generic/Ramp/Ramp5	465.267	465.267	Ramp5	Float8	
Sim_Generic/Ramp/Ramp4	158.320	158.320	Ramp4	Float8	

Execution Scheduling:
● Timer ○ Schedule
1 second(s)
Data source:
MySQL
Table name:
ramp_historical
☑ Automatically create table

```
SELECT * FROM ramp_historical
```

☑ Limit SELECT to: 1000 rows

ramp_historical_...	Ramp0	Ramp1	Ramp2	Ramp3	Ramp5	Ramp4	t_stamp
1	794.84	96.13	169.84	304.195	496.775	396.13	2019-06-18 10:2
2	808.187	6.14	-16.813	19.21	5.117	6.14	2019-06-18 10:2
3	821.52	16.14	-3.48	34.21	13.45	16.14	2019-06-18 10:2
4	834.867	26.15	9.867	49.225	21.792	26.15	2019-06-18 10:2
5	848.2	36.15	23.2	64.225	30.125	36.15	2019-06-18 10:2
6	861.547	46.16	36.547	79.24	38.467	46.16	2019-06-18 10:2
7	874.88	56.16	49.88	94.24	46.8	56.16	2019-06-18 10:2
8	888.227	66.17	63.227	109.255	55.142	66.17	2019-06-18 10:2
9	901.56	76.17	76.56	124.255	63.475	76.17	2019-06-18 10:2
10	914.92	86.19	89.92	139.285	71.825	86.19	2019-06-18 10:2

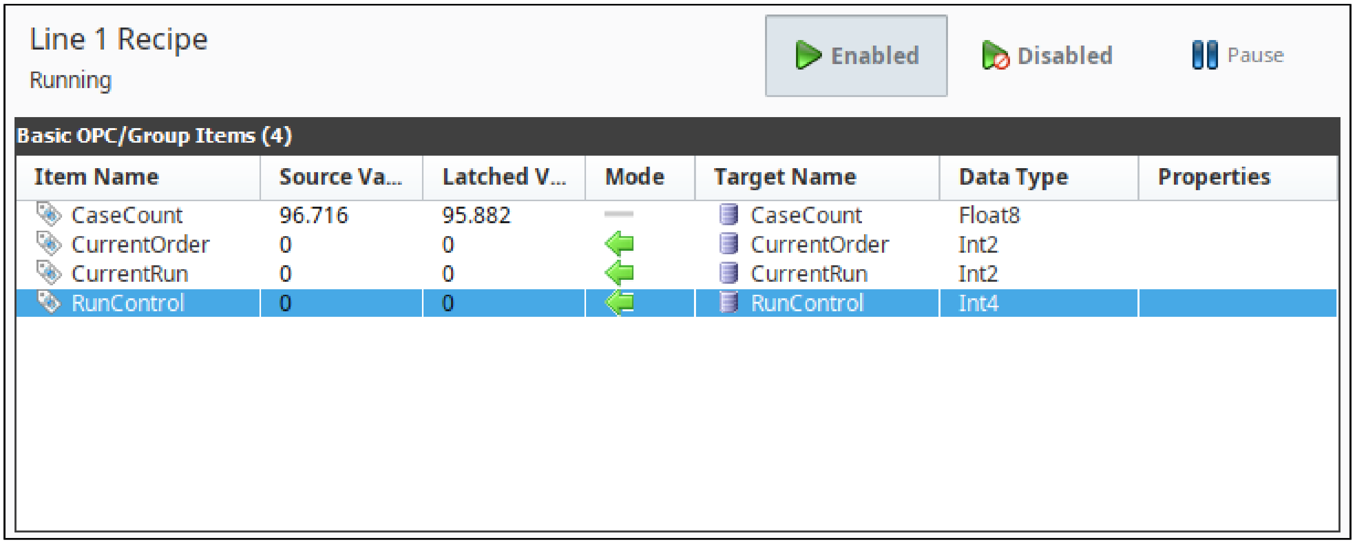
268 rows fetched in 0.013s

SELECT * FROM Tank_Histc
SELECT * FROM ramp_histc

Recipe Management and Synchronization

Transaction groups can furnish devices connected to Ignition with information stored in a database, based on a parameter from the device or elsewhere. Typically a Standard Group would be used for this application, but Block and Stored Procedure groups can accomplish this as well. See the Recipe example for more details.

More generally, Standard and Block groups can be used to synchronize device or tag data with a database table on a timer or schedule, allowing the database to act as a go-between for another system and a target device, or for the database to reflect realtime values for specified data points.

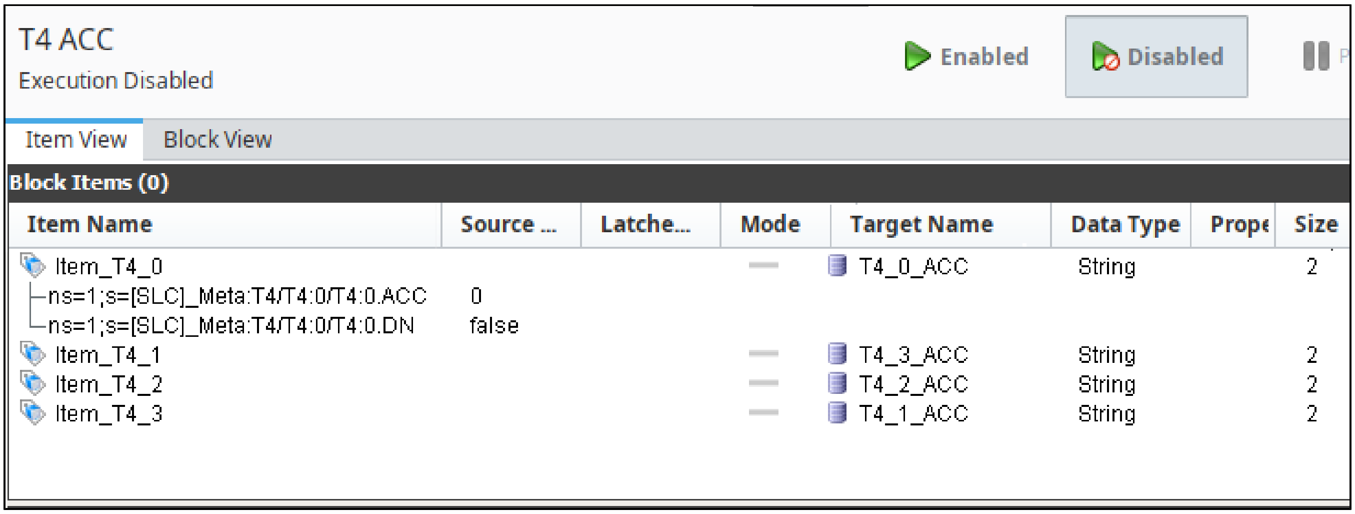


Basic OPC/Group Items (4)

Item Name	Source Va...	Latched V...	Mode	Target Name	Data Type	Properties
CaseCount	96.716	95.882	—	CaseCount	Float8	
CurrentOrder	0	0	←	CurrentOrder	Int2	
CurrentRun	0	0	←	CurrentRun	Int2	
RunControl	0	0	←	RunControl	Int4	

Block Data Transfers

The Block Group supports the transfer of entire arrays of data to and from the database.



T4 ACC

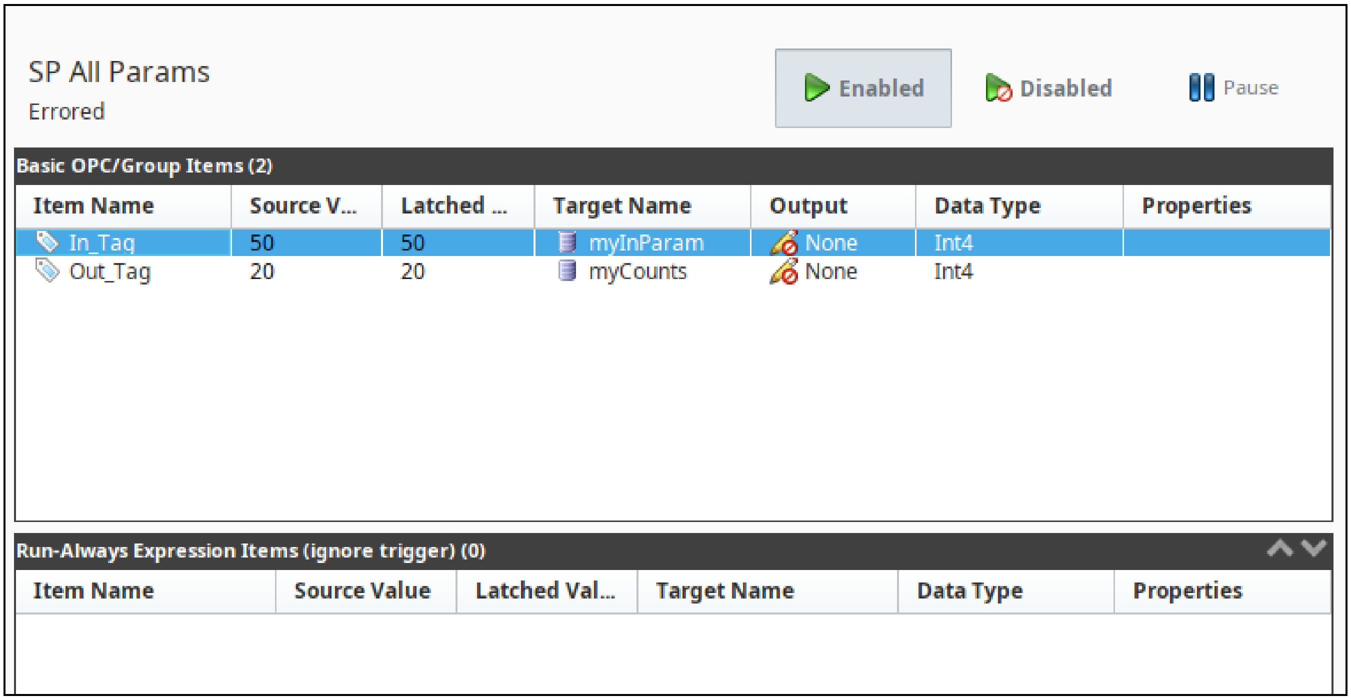
Execution Disabled

Block Items (0)

Item Name	Source ...	Latche...	Mode	Target Name	Data Type	Prope	Size
Item_T4_0			—	T4_0_ACC	String		2
ns=1;s=[SLC]_Meta:T4/T4:0/T4:0.ACC	0						
ns=1;s=[SLC]_Meta:T4/T4:0/T4:0.DN	false						
Item_T4_1			—	T4_3_ACC	String		2
Item_T4_2			—	T4_2_ACC	String		2
Item_T4_3			—	T4_1_ACC	String		2

Stored Procedures

The Stored Procedure Group allows you to use group items as inputs and outputs for your existing Stored Procedures, allowing query specifics to be managed external to Ignition.



SP All Params

Errored

Basic OPC/Group Items (2)

Item Name	Source V...	Latched ...	Target Name	Output	Data Type	Properties
In_Tag	50	50	myInParam	None	Int4	
Out_Tag	20	20	myCounts	None	Int4	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Value	Latched Val...	Target Name	Data Type	Properties

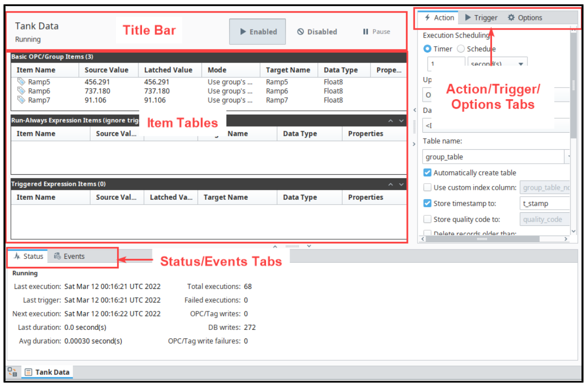
In This Section ...

Understanding Transaction Groups

Transaction Group Workspace

Transaction Groups are edited through the Ignition Designer. When a group is selected, you are presented with the transaction group workspace. The workspace is broken into several parts:

- **Title bar** - Shows the name of the currently selected group, as well as options to set it as Enabled or Disabled, and to Pause, if it's currently executing.
- **Item tables** - Shows all of the items configured in the selected group. Many settings can be modified directly through the display, the rest by double-clicking the item or selecting **Edit** in the context menu.
- **Action / Trigger / Options tabs** - Defines how and when a group executes. Holds most of the options that apply to the group in general, such as the update rate, and which data connection it uses.
- **Status / Events tabs** - Provides information about the executing group, including the most recent messages that have been generated.



Basic OPC/Group Items (3)

Item Name	Source Value	Latched Value	Mode	Target Name	Data Type	Prope...
Ramp5	456.291	456.291	Use group's ...	Ramp5	Float8	
Ramp6	737.180	737.180	Use group's ...	Ramp6	Float8	
Ramp7	91.106	91.106	Use group's ...	Ramp7	Float8	

Run-Always Expression Items (ignore trig

Item Name	Source Val...		Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source Val...	Latched Va...	Target Name	Data Type	Properties

Execution Scheduling

● Timer ○ Schedule

1 second(s)

Table name: group_table

☑ Automatically create table

□ Use custom index column: group_table_nd

☑ Store timestamp to: t_stamp

□ Store quality code to: quality_code

□ Delete records older than:

Running

Last execution: Sat Mar 12 00:16:21 UTC 2022
Last trigger: Sat Mar 12 00:16:21 UTC 2022
Next execution: Sat Mar 12 00:16:22 UTC 2022
Last duration: 0.0 second(s)
Avg duration: 0.00030 second(s)

Total executions: 68
Failed executions: 0
OPC/Tag writes: 0
DB writes: 272
OPC/Tag write failures: 0

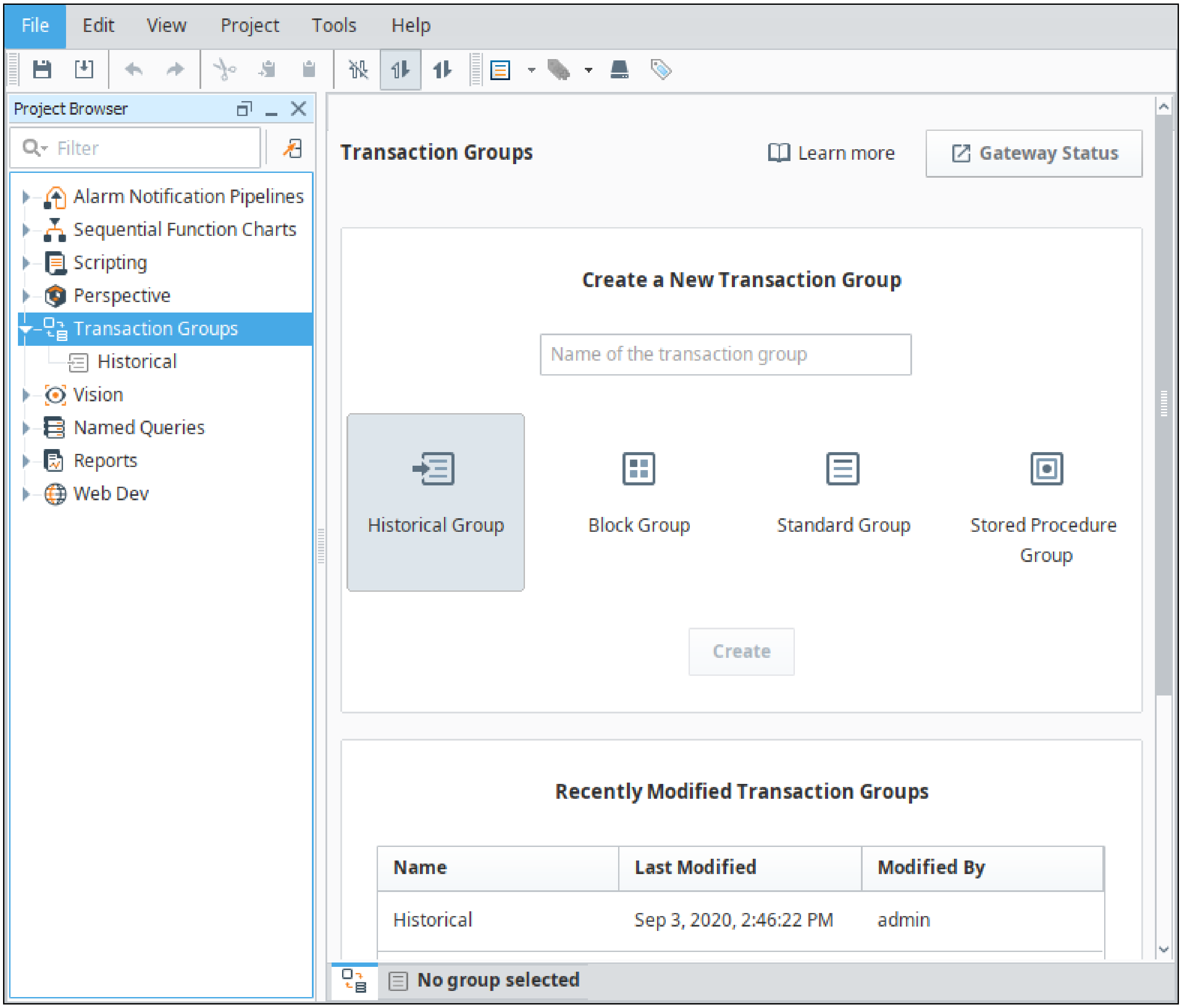
On this page ...

Transaction Group Introduction

Watch the Video

Welcome Page

The Transaction Group Welcome page allows you to create any one of the four types of transaction groups. Each one of the transaction groups is basically a template to help you get started creating your transaction group. Once you select a Transaction Group, enter a name, and press 'create', and the specific transaction group template will open. All you have to do next is enter the tags for the values you want to collect. The Transaction Group Welcome tab will show you any recently modified transaction groups along with the date it was modified and who modified it. You can even double click on a recently modified chart and open it.



Name	Last Modified	Modified By
Historical	Sep 3, 2020, 2:46:22 PM	admin

Items

Each Item (Tag) in the Transaction Group consists of several properties, but the key properties are the Source/Latched Values and Target Name.

Source and Latched Value

The Source Value will be the value of the items source. This can be something like a Tag or a direct OPC Item if writing to the database, but can also be the value pulled from the database if in DB to OPC mode. This value can change in between executions, depending on the source type. When the source is a Tag, it will update as the Tag updates, depending on how the Tag Group for the Tag is set. However, if the source is an OPC Item, it will update only when the group executes, unless the OPC subscription rate is overridden in the group.

The Latched value will be the value that was written at execution. This can be the value that gets written to the database on execution in OPC to DB mode, or it can be the value that gets written to the Tag in DB to OPC mode. The value will only change on execution of the group.

Target Name

In most cases, the Target Name is a column on the database table the Transaction Group is associated with. However, it is possible to have the Target Name 'Read-only'. When set to 'Read-only' the value of the item will not be tied to any columns in the database, but is still visible from the Transaction Group and can be used as a trigger.

Enabling Group Execution

In order for groups to be evaluated, they must first be enabled. This is done by selecting **Enabled** in the group title bar, and then saving the project. The group executing can be stopped by reversing the procedure and selecting **Disabled** before saving. If you want to quickly and temporarily stop the group's evaluation, toggle the **Pause** button. This will prevent execution until the group is enabled again, or until the system is restarted.

Note: Transaction Groups exist in a project, but they execute in the Gateway space. This means that once your groups are enabled, they do not need (or use) Vision clients or Perspective sessions in order to run.

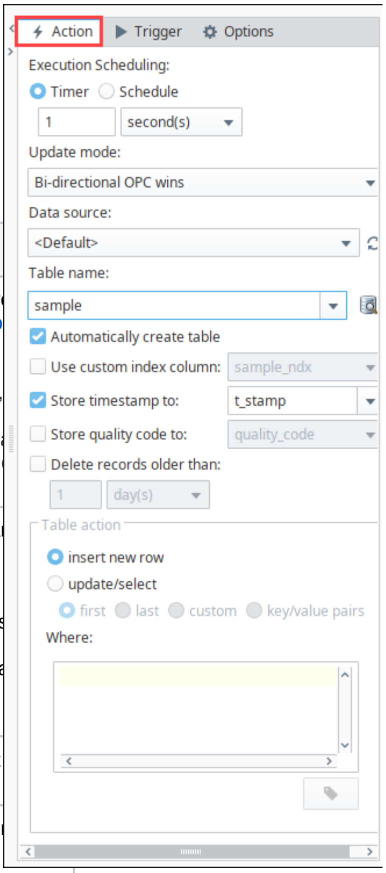
Editing Group Settings

Group settings may be modified at any time, regardless of whether or not the group is executing. Modifications will be applied when the project is saved, and the group will be started or stopped as required. Some changes such as modifying items may cause features like live values to appear to be incorrect. It is therefore important to notice the modified icon that appears next to the group, and to save often. If you would prefer to stop the group before making edits you can simply pause the group. Execution will begin again after the project is saved.

Action Settings

The action settings of a Transaction group define how often the group will be evaluated, as well as important settings that apply to the group as a whole. They are found on the **Action** tab, the first of the tabs on the right side of the Transaction Group workspace.

The Action settings vary for the different types of Transaction Groups, but a few settings are common to most of them:



Execution Scheduling:

● Timer ○ Schedule

1 second(s)

Update mode: Bi-directional OPC wins

Data source: <Default>

Table name: sample

☑ Automatically create table

☐ Use custom index column: sample_ndx

☑ Store timestamp to: t_stamp

☐ Store quality code to: quality_code

☐ Delete records older than: 1 day(s)

Table action

● insert new row

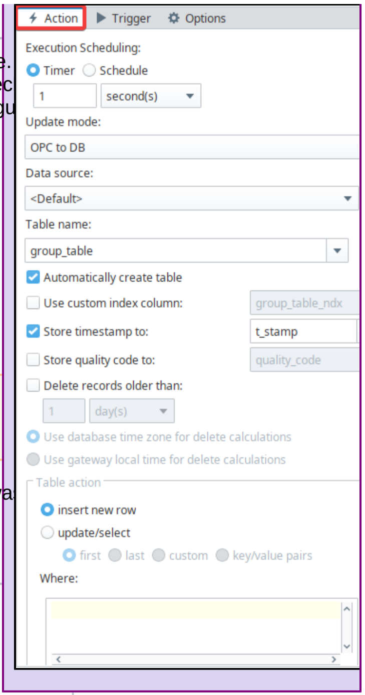
○ update/select

● first ○ last ○ custom ○ key/value pairs

Where:

Setting	Description
Execution scheduling	Despite the name, determines how often the group is *evaluated*. For a number of reasons, the gr may not execute during the evaluation. The most common reason is the trigger, but see Executio Cycle below for more possible reasons why evaluation will exit. - **Timer** - specifies the OPC Tag subscription rate for the OPC Tags. It can run at millisecond, second, minute, hour, or day rates. - **Schedule** - is a specified start time on the update **Rate**. Set a list of time (or time ranges) th group should evaluate at. If the pattern specified includes a time range, a rate must be provi
Update mode	For groups that support it, sets the default for how items are compared to their targets. Options a - OPC to DB - Only read from the OPC server and write to the database. - DB to OPC - Only read from the database and write to the OPC Server. - Bi-directional OPC wins - Read and Write to both the database and OPC Server. On group s write OPC values to the database. - Bi-directional DB wins - Read and Write to both the database and OPC Server. On group sta write database values to OPC items.
Data source	The database connection name the group should use. Can be **Default**, which will use the default connection for the project.
Table name	Name of the table in the database that the group should interact with (reading or writing, dependi the **Update mode** and individual item **Mode** settings). The tables listed in this dropdown are determined by the **Data source** property. This setting allows you to type arbitrary names into it. If you type the name of a database table that doesn't exist, and the **Automatically create table** setting is enabled, then the group will attempt to create the database table on start.
Automatically create table	If enabled, the Transaction Group will attempt to create a database table once the group starts running, assuming one doesn't already exist as determined by the **Table name** setting. If the table already exists, then nothing happens.
Use custom index	If left disabled, the group will attempt to add an index column to the database table when the group starts executing. If enabled, the group will use the column selected in the adjacent dropdown, or create a new column if you type in a column name that doesn't exist on the table (requires the **Automatically create table** setting to be enabled).
Store timestamp	If enabled, will attempt to store a timestamp value to the column specified in the adjacent dropdown. If you type in a column name that doesn't exist on the table, the group will attempt to create the column on start, assuming the **Automatically create table** setting.
Store quality	Stores an aggregate quality for the group along with the regular data. The aggregate quality is a bit-wise AND of the qualities of the items in the group.

Editor notes are only visible to logged in users
From Brandon: I was originally going to get rid of this screenshot since IGN-3350 was reverted but it might be prudent to leave this here for now in case the change goes through in the future. See IGN-6310 for the original ticket information and IGN 6316 for further solution research



Action | Trigger | Options

Execution Scheduling:
● Timer ○ Schedule
1 second(s)

Update mode:
OPC to DB

Data source:
<Default>

Table name:
group_table

☑ Automatically create table
□ Use custom index column: group_table_ndx
☑ Store timestamp to: t_stamp
□ Store quality code to: quality_code
□ Delete records older than: 1 day(s)
● Use database time zone for delete calculations
○ Use gateway local time for delete calculations

Table action
● insert new row
○ update/select
● first ○ last ○ custom ○ key/value pairs
Where:

<table>
<tr><td>code</td><td></td></tr>
<tr><td>Delete records older than</td><td>

If enabled, and the group is running, this setting will make the group delete older rows in the table. Options are minute(s), day(s), month(s), and year(s). While enabled, a task will be created to chec the database tables for older records. The rate this task occurs is determined by frequency configu on this property.

Frequency	Task execution rate
Less than an hour	Every minute
Between one and twelve hours	Every 15 minutes
Over twelve hours	Every hour

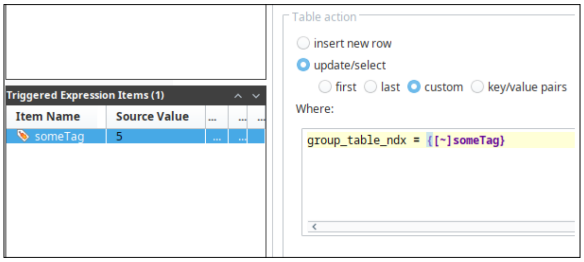
The following feature is new in Ignition version **8.1.19**
Click here to check out the other new features

Transaction Groups using this property will use timestamps instead of the row index the record wa inserted at when pruning.

</td></tr>
<tr><td>Table Action</td><td>

Defines which row will be targeted by the group.

- **insert new row**
- **update/select** - allows you to target specific rows in the database table. Options are:
  - **first row** - the group always executes against the first row.
  - **last row** - the group always executes against the last row
  - **custom** - allows you to write a custom where clause to determine which row should be targeted. Uses the **Where** text area. The custom clause can use references to values of items in the group.



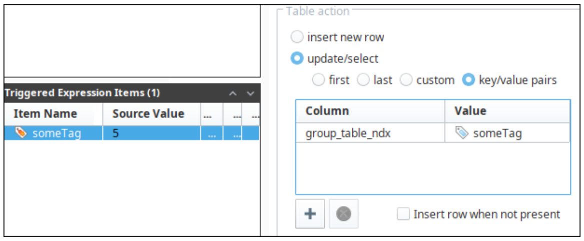
Triggered Expression Items (1)

Item Name	Source Value
someTag	5

Table action
○ insert new row
● update/select
○ first ○ last ● custom ○ key/value pairs
Where:

```
group_table_ndx = {[~]someTag}
```

  - **key/value pairs** - Provides dropdowns for both a column and a item in the group, allowing the group to target a single row in a table based on the item's value. For example, if a value of 5 is used in conjunction with the "group_table_ndx" column in the database table, rows where group_table_ndx has a value of 5 are targeted when the group executes.



Triggered Expression Items (1)

Item Name	Source Value
someTag	5

Table action
○ insert new row
● update/select
○ first ○ last ○ custom ● key/value pairs

Column	Value
group_table_ndx	someTag

□ Insert row when not present

Additional conditions can be added or removed with the Add or Delete buttons at the bottom of the Table Action settings.

</td></tr>
</table>

Group Update Rate

Groups generally work on a timer. They are set to run at a certain rate. As they are running at that certain rate, they then check the rest of the settings. If the trigger conditions pass, the group is executed fully.

The Execution Schedule controls the rate at which the transaction group executes. On the Action tab of a group you selected, under Execution Scheduling, there are two options: **Timer** and **Schedule**. Timer, executes the group at a certain rate. Schedule, executes the group at specific times. When the Schedule option spans across a period of time, you must specify the rate at which the group executes during that time.

Group Update Rate

Watch the Video

Timer

The Timer acts as the heartbeat of the transaction group and is evaluated at the provided rate. It can run at millisecond, second, minute, hour, or day rates. The Timer specifies the OPC Tag subscription rate for the OPC Tags. When a Timer is running the transaction group it first analyzes the Tags inside the **Basic OPC/Group Items** section of the transaction group. Then it looks at the trigger configuration and evaluates for Tag changes. Then it evaluates the specific trigger conditions and decides to execute on a trigger. Depending on the trigger settings, full execution may not occur, but the trigger will at least be evaluated at this rate. If the triggered condition is true, the transaction group proceeds to the **Triggered Expression Items** section of the transaction group. Only after this flow is complete, will the transaction group interact with the database, and for example, insert the Tag values into the database.

Schedule

An important difference between the Timer and the Schedule options is that the schedule option will automatically align to the specified start time on the update rate. With Schedule mode, you are providing a list of time (or time ranges) that the group should run at. If the pattern specified includes a time range, a rate must be provided, and the group will evaluate as in timer mode during that period.

The schedule is specified as a comma separated list of times or time ranges. You may use the following formats:

- 24-hour times. "8:00, 15:00, 21:00", for execution at 8am, 3pm, and 9pm.
- 12-hour with am/pm (if not specified, "12" is considered noon): "8am, 3pm, 9pm"
- Ranges, "8am-11am, 3pm-5pm"
- Ranges that span over midnight, such as "9pm - 8am"

When using ranges, the execution times will be aligned to the start time. For example, if you specify a schedule of "9am - 5pm" with a rate of "30 minutes", the group will evaluate at 9, 9:30, 10, etc., regardless of when it was started. This is a useful difference compared to the Timer mode, which runs based on when the group was started. For example, if you want a group that runs every hour, on the hour, you could specify a 1 hour rate with a range of "0-24."

Execution Cycle

All of the Transaction Groups follow a similar execution cycle. The flow may differ based on the group's configuration, but the general cycle is always the same.

1. Timer determines it is time to evaluate.
2. Is the group paused? If so, end the cycle.
3. Is the Gateway part of a redundant pair? If so, is it active? If not active, end the cycle. Groups only execute on the active node.
4. Evaluate run-always items: OPC items, Tag references, and expression items set to ignore the trigger (or items placed in the run always section of the Configuration window).
5. Is trigger set/active? If there is a trigger defined, but it is not active, end the cycle.
6. Evaluate "triggered" Expression items.
7. If applicable, read values from the database.
8. Compare items with their targets.
9. Execute any writes.
10. Report alarms configured on the executed items
11. Acknowledge the trigger, if applicable.
12. Write handshake value, if applicable.

If an error occurs at any stage besides the last stage, the cycle will end and a failure handshake will be written, if configured. The group will attempt execution again after the next update rate period.

When the "Bypass Store and Forward System" option is false, writing to a database from the Transaction Group will result in a successful execution if the database write enters the Store and Forward Pipeline.

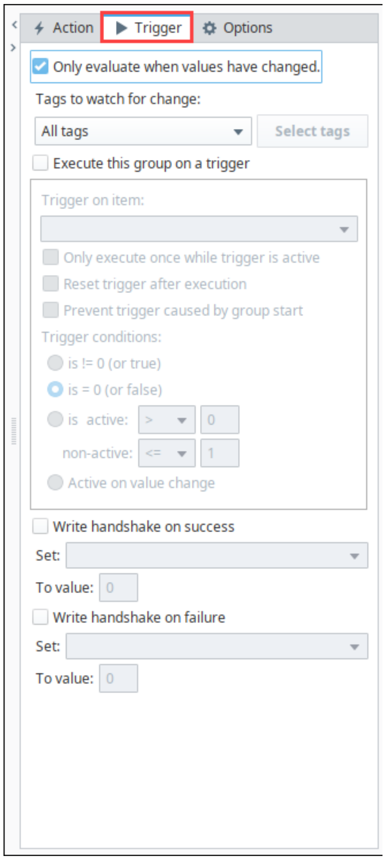
Caution: If the group errors due to a bad database connection, it will need to be manually restarted once the database connection is brought back.

Trigger Settings

The Trigger tab contains settings that modify how the group executes. The outcome of an execution is handled in the handshake section of the trigger section of the transaction group.

The table below is a list of Trigger and Handshake settings.

Setting	Description
Only evaluate when values have changed	Enabling this setting will cause the group to evaluate only if values **or** Tag qualities have changed. If the values have not changed, it will exit the evaluation. You have the option to monitor all Run-Always items in the group, or only specific ones. • **Tags to watch for change** - Select either all Tags or one or more Tags in order to monitor for value changes. Select 'all Tags' or 'Custom,' and select the Tag(s) from the dropdown.
Execute this group on a trigger	Enables a trigger on a specific item in the group. The trigger item can be any Run-Always item, such as an OPC item, Tag reference, or an Expression item set to "Run-Always" mode. **Note:** When using triggers in a Block Group, the intended trigger will need to be under **Basic OPC/Group Items** instead of **Block Items** order for the trigger to show in the **Trigger on item** dropdown list. • **Trigger on item** - select the item time you want to use as the trigger.
Only execute once while trigger is active	The group will only execute once when the trigger goes into an active state, and will not execute again until the trigger goes inactive first. If unselected, the group will execute each cycle while the trigger conditions evaluate to true.
Reset trigger after execution	If using the ">0" or "=0" trigger modes, the trigger can be set to write an opposite value after the group has executed successfully. This is useful for relaying the execution back to the PLC.
Prevent trigger caused by group start	If selected, the group will not execute if the trigger is active on the first evaluation of the group after enabling the group. Selecting this option will prevent initial executions caused by system restarts, or reenabling the group.
Trigger conditions	Set any of the following trigger conditions: • is !=0 (or true) • is =0 (or false) • is active or non-active, which causes the group to execute if the trigger value matches the **is active** condition. • Active on value change, which will cause the group to execute if the trigger changes value at all.
Write handshake on success	Set the item and the value you want to see when the group executes successfully.
Write handshake on failure	Set the item and the value you want to see when an error prevents the group execution.



☑ Only evaluate when values have changed.

Tags to watch for change: All tags

□ Execute this group on a trigger

Trigger on item:

□ Only execute once while trigger is active

□ Reset trigger after execution

□ Prevent trigger caused by group start

Trigger conditions:

○ is != 0 (or true)

● is = 0 (or false)

○ is active: > 0

non-active: <= 1

○ Active on value change

□ Write handshake on success

Set:

To value: 0

□ Write handshake on failure

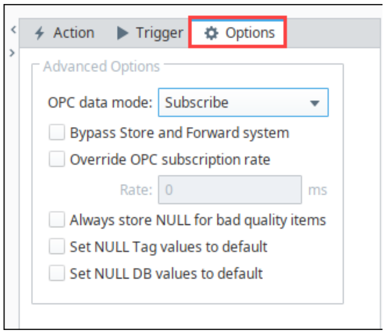
Set:

To value: 0

To learn more about configuring Transaction Groups with the different trigger options, refer to the Trigger Options page.

Advanced Settings

Transaction Groups offer several advanced settings that affect how execution occurs. These settings can be found under the **Options** tab for a group. The table below describes the Advanced settings.



Advanced Options

OPC data mode: Subscribe

□ Bypass Store and Forward system

□ Override OPC subscription rate

Rate: 0 ms

□ Always store NULL for bad quality items

□ Set NULL Tag values to default

□ Set NULL DB values to default

<table>
<tr><th>Setting</th><th>Description</th></tr>
<tr><td>OPC Data Mode</td><td>Modifies how the group receives data from OPC.
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td><b>Subscribe</b></td><td>Data points are registered with the OPC server, and data is received by the group <b>on-change</b>. This is the default setting and generally offers the best</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td><table>
<tr><td></td><td>performance, as it reduces unnecessary data flow and allows the OPC server to optimize reads.

<b>Note:</b> Data is received by the group asynchronously, meaning that it can arrive at any time. When the group executes, it "snapshots" the last values received and uses those during evaluation. If some values arrive after execution begins, they will not be used until the following execution cycle.</td></tr>
<tr><td><b>Read</b></td><td>Each time the group executes it will first read the values of OPC items from the server. This operation takes more time and involves more overhead than subscribed evaluation, but ensures that all values are updated together with the latest values. It is therefore commonly used with batching situations, where all of the data depends on each other and must be updated together. It's worth noting that when using an OPC item as the trigger, the item will be subscribed, and the rest of the values read when the trigger condition occurs.</td></tr>
</table></td></tr>
<tr><td>Bypass Store and Forward System</td><td>This setting is only applicable to groups that insert rows into the database. Causes groups to target the database directly instead of going through the store-and-forward system. If the connection becomes unavailable, the group will report errors instead of logging data to the cache.</td></tr>
<tr><td>Override OPC subscription rate</td><td>Specifies the rate at which OPC items in the group will be subscribed. These items are normally subscribed at the rate of the group, but by modifying this setting it is possible to request updates at a faster or slower rate.</td></tr>
<tr><td>Always store NULL for bad quality items</td><td>With this option set to True, it will force the group to store a NULL value when the item has a bad quality, instead of writing the bad quality value.</td></tr>
<tr><td>Set NULL Tag values to default</td><td>If a NULL is read from the Tag, it will instead use a default value to write to the database, depending on the type. This can prevent errors for database columns that do not accept NULL values. The default values are the same as the table above.</td></tr>
<tr><td>Set NULL DB values to default</td><td>If a NULL is read from the database, it will instead use a default value to write to the Tag, depending on the type. This can prevent errors for OPC Tags that do not accept NULL values. Not available in a Historical Group.

Enabling the Set DB Values to Default setting on Block Groups will clear the latched value, setting the item to a default if the corresponding database value is Null.

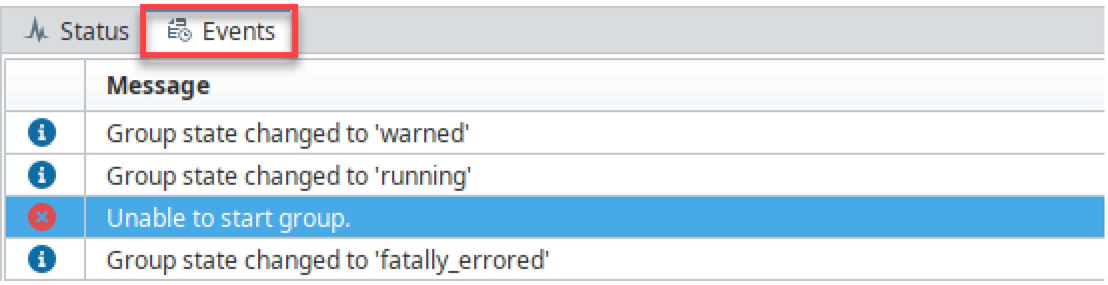
<table>
<tr><th>Type</th><th>Default Value</th></tr>
<tr><td>Byte</td><td>0</td></tr>
<tr><td>Short</td><td>0</td></tr>
<tr><td>Integer</td><td>0</td></tr>
<tr><td>Long</td><td>0</td></tr>
<tr><td>Float</td><td>0.0</td></tr>
<tr><td>Double</td><td>0.0</td></tr>
<tr><td>Boolean</td><td>FALSE</td></tr>
<tr><td>String</td><td>'' (Empty Sting)</td></tr>
<tr><td>Date/Time</td><td>Current Date/Time</td></tr>
<tr><td>Dataset</td><td>[0x0] (Empty Dataset)</td></tr>
<tr><td>Array</td><td>[] (Empty Array)</td></tr>
</table></td></tr>
</table>

Troubleshooting

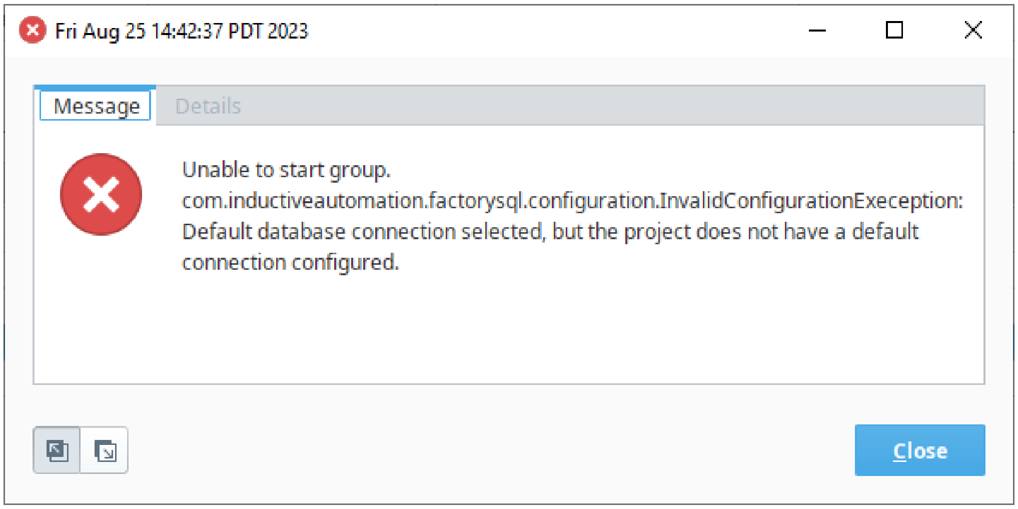
It may be helpful when troubleshooting or testing Transaction Groups to increase the default threadpool count. Refer to the Gateway Configuration File Reference - Threadpool Counts for more information.

Events Tab

In the Events tab, the list of events will have a brief summary of the occurrence. Double-click an event to display a pop-up with a more detailed description.



	Message
ℹ	Group state changed to 'warned'
ℹ	Group state changed to 'running'
✖	Unable to start group.
ℹ	Group state changed to 'fatally_errored'

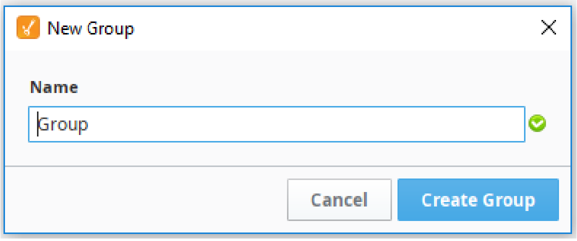

Fri Aug 25 14:42:37 PDT 2023

Unable to start group.
com.inductiveautomation.factorysql.configuration.InvalidConfigurationExeception:
Default database connection selected, but the project does not have a default connection configured.

Creating a Transaction Group

This example demonstrates how to configure a transaction group, specifically a Historical Group. However, the process of creating any transaction group type is very similar, especially so in the case of a standard group. The Transaction Group Examples section contains more examples.

1. Click on the **Transaction Groups** in the Project Browser to switch the Designer's workspace to the Transaction Group workspace.
2. In the Project Browser, right click on **Transaction Groups > New Transaction Group** to make a New Historical Group. Name the group 'Group.'



Name: Group

3. Browse your OPC device and drag some OPC Tags to the **Basic OPC/Group Items** section. The group starts out 'Disabled' by default.

INDUCTIVE UNIVERSITY

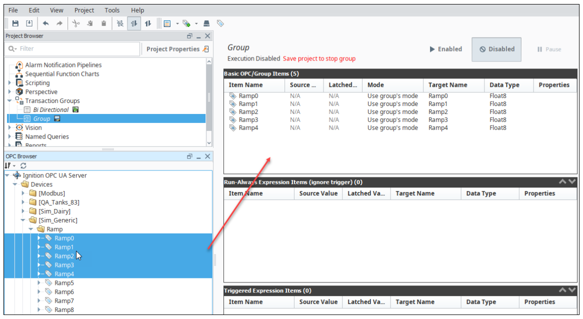
Basic Historical Group

Watch the Video

INDUCTIVE UNIVERSITY

Realtime Group

Watch the Video



Group

Execution Disabled Save project to stop group

Basic OPC/Group Items (5)

Item Name	Source ...	Latched...	Mode	Target Name	Data Type	Properties
Ramp0	N/A	N/A	Use group's mode	Ramp0	Float8	
Ramp1	N/A	N/A	Use group's mode	Ramp1	Float8	
Ramp2	N/A	N/A	Use group's mode	Ramp2	Float8	
Ramp3	N/A	N/A	Use group's mode	Ramp3	Float8	
Ramp4	N/A	N/A	Use group's mode	Ramp4	Float8	

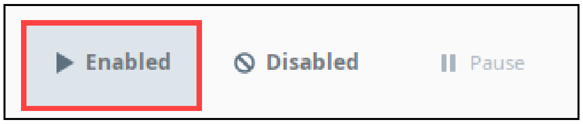
Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Value	Latched Va...	Target Name	Data Type	Properties

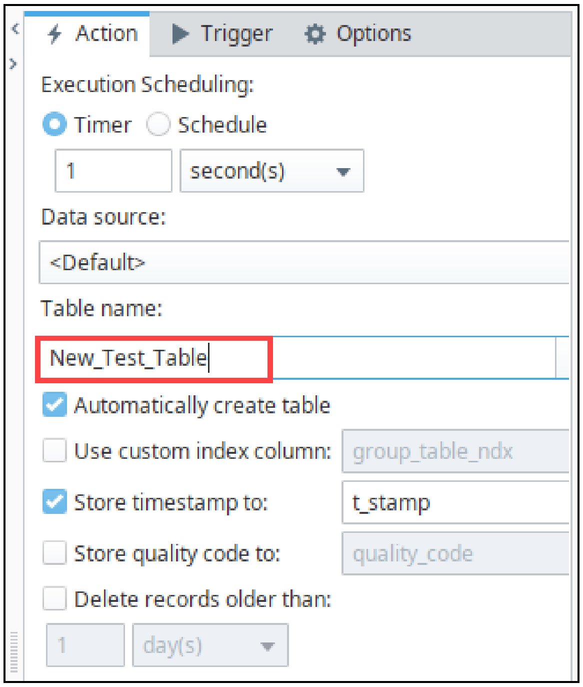
Triggered Expression Items (0)

Item Name	Source Value	Latched Va...	Target Name	Data Type	Properties

4. Save your project.
5. Click the **Enabled** button above the item tables to enable logging.



6. Go to the **Action** tab and change the **Table Name**. For the example, we gave it the name "New_Test_Table."
   At this step, your group only exists in the Designer.



Execution Scheduling:

● Timer ○ Schedule

1 second(s)

Data source:

<Default>

Table name:

New_Test_Table

☑ Automatically create table

□ Use custom index column: group_table_ndx

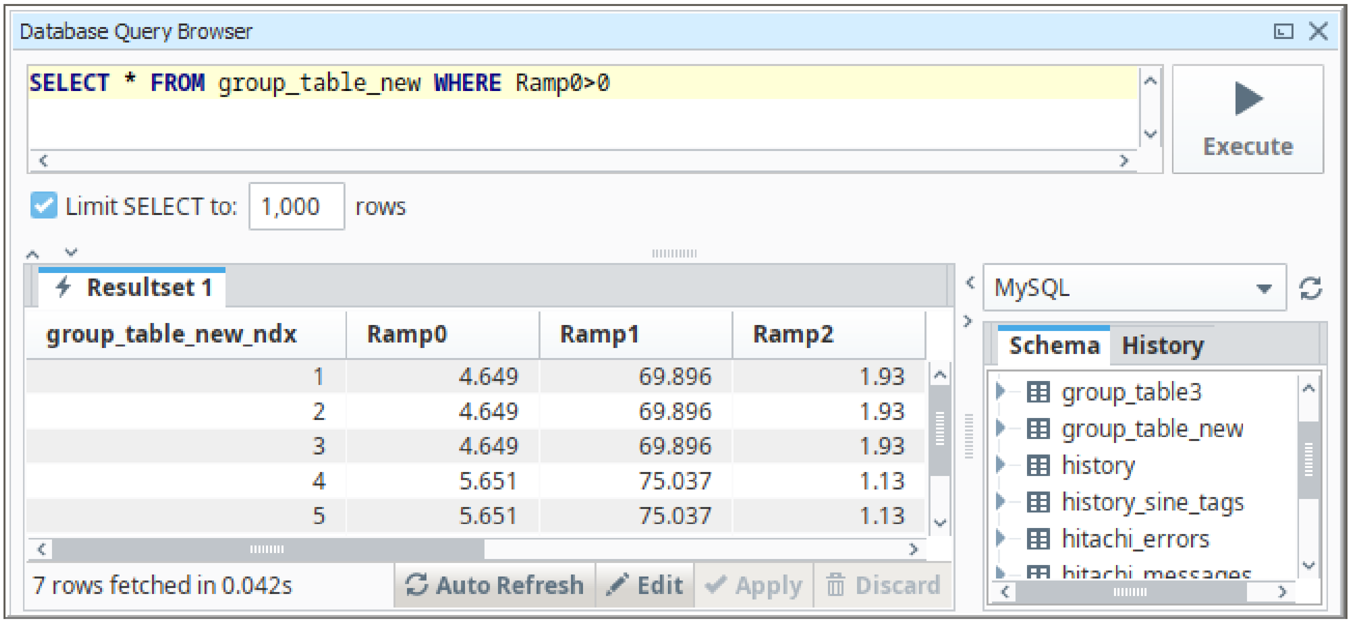
☑ Store timestamp to: t_stamp

□ Store quality code to: quality_code

□ Delete records older than:

1 day(s)

7. **Save** your project to start the group. Your group is now running and logging data to your database connection.
8. To see the data, you can use the Ignition Designer's built-in Database Query Browser. The easiest way to do this is to click on the **Database** icon next to your group's Table Name field. The Query Browser is a convenient way to directly query your database connection without leaving the Ignition Designer. Of course, you can also use any query browser tools that came with your database.



```
SELECT * FROM group_table_new WHERE Ramp0>0
```

☑ Limit SELECT to: 1,000 rows

group_table_new_ndx	Ramp0	Ramp1	Ramp2
1	4.649	69.896	1.93
2	4.649	69.896	1.93
3	4.649	69.896	1.93
4	5.651	75.037	1.13
5	5.651	75.037	1.13

7 rows fetched in 0.042s

Types of Groups

The SQL Bridge Module provides four different types of Transaction Groups that you can use in your projects. Each of these different types of groups vary in their operation and use for data logging and database to PLC synchronization.

On this page ...

Standard Group

The Standard Group contains items, which may be mapped to columns in the group's linked database table, or used internally for features such as triggering or handshakes. Items that are mapped to the database target a specific column of a single specific row, chosen according to the group settings. Values can flow from the items into the database, from the database to the items, or bidirectionally, allowing the value of the database and the item will be synchronized.

The group may also insert new rows instead of updating a specific row, similar to the Historical Group.

Group Settings

The Standard Group uses a timer-based execution model shared by all groups, and the normal trigger settings. Additionally, there are several settings specific to the group type:

- **Automatically create table** - If the target table does not exist, or does not have all of the required columns, it will be created/modified on group startup. If not selected and the table doesn't match, an error will be generated on startup.
- **Use custom index column** - If selected, you may enter any column name to hold the index. If unselected, the table index will be named <table name>_ndx.
- **Store timestamp to** - Specifies whether or not to store a timestamp with the record, and the target column. The timestamp will be generated by the group during execution. For groups that update a row, the timestamp will only be written if any of the values in the group are also written.
- **Store quality code to** - If selected, stores an aggregate quality for the group to the specified column. The aggregate quality is the combined quality of all of the items that write to the table. For more information about quality values, see Data Quality.
- **Delete records older than** - If selected, records in the target table will be deleted after they reach the specified age. This setting is useful for preventing tables from growing in an unbounded manner, which can cause disk space and performance problems over time.

Table Action

This section details how the group interacts with the table on each execution. The group can insert a new row, or select/update the first, last or a custom record.

- **Insert New Row** - This option will make the group insert a new record into the database every time the group executes. This is the forced behavior of the Historical Group.
- **Update / Select** - This option will either update or select from matching rows based on the option selected below it. The **Update Mode** property above determines whether an update (OPC to DB), select (DB to OPC), or both (Bi-directional) are used when the group executes.
- **First** - Use the first row in the table. It is not recommended to use this option unless the order of the data in the table is guaranteed.
- **Last** - Use the last row in the table. This is commonly used when another group (or another program) is inserting new rows for us, and we always want to update the most recent record.
- **Custom** - A custom update clause is essentially the WHERE clause of the SQL query that will be generated to read and write the group data. This usually contains a reference to a Tag in the group. IE: *column_name = {[~]item_name}*
- **Key/Value Pairs** - Used to inject dynamic values in order to create a WHERE clause for you. The table below this option will allow you to enter column names and link them to values (usually Tags in the group). This option also has the ability to Insert a new row with the current key/value pair if it was not found.

Typical Uses

Standard Groups can be used any time you want to work with a single row of data. This can include:

- **Historical logging** - Set the group to insert new records, and log data historically either on a timer, or as the result of a trigger. Flexible trigger settings and handshakes make it possible to create robust transactions.
- **Maintain status tables** - Keep a row in the database updated with the current status values. Once in the database, your process data is now available for use by any application that can access a database, dramatically opening up possibilities.
- **Manage recipes** - Store recipe settings in the database, where you have a virtually unlimited amount of memory. Then, load them into the PLC by mapping DB-to-OPC using a custom where clause with an item binding in order to dynamically select the desired recipe.
- **Sync PLCs** - Items in the group can be set to target other items, both for one-way and bidirectional syncing. By adding items from multiple PLCs to the group, you can set the items

of one PLC to sync with the others. By creating expression items that map from one PLC item to the other, you can manipulate the value before passing it on.

Historical Group

The Historical Group inserts records of data into a SQL database, mapping items to columns. Full support for triggering, expression items, hour & event meters and more means that you can also set up complex historical transactions. Unlike the Standard Group, the Historical Group cannot update rows, only insert. It also cannot write back to items (besides trigger resets and handshakes).

Group Settings

The settings of the Historical Group are identical to the settings in the Standard Group, but limited to inserting rows.

Typical Uses

- **Basic Historical Logging** - Recording data to a SQL database gives you incredible storage and querying capabilities, and makes your process data available to any application that has DB access.
- **Shift Tracking** - Use an expression item to track the current shift based on time, and then trigger off of it to record summary values from the PLC. Use a handshake to tell the PLC to reset the values.

Block Group

Block Groups instead allow you to store your data in a tall format. They allow you to create a unique type of item, called a **Block Item**, which represents an ordered list of values to store within a column for each execution.

INDUCTIVE UNIVERSITY

Block Group Type

Watch the Video

General Description

A Block Group contains one or more block items. Each block item maps to a column in the group's table, and then defines any number of values (OPC or SQLTag items) that will be written vertically as rows under that column. The values may be defined in the block item in two modes. The first, List mode, lets a list of value-defining items to be entered. These value items may either be OPC items, Tag items, or static values. The second mode, Pattern mode, can be useful when OPC item paths or Tag paths contain an incrementing number. You may provide a pattern for the item's path, using the wildcard marker {?} to indicate where the number should be inserted.

Block groups are very efficient, and can be used to store massive amounts of data to the database (for example, 100 columns each with 100 row -10,000 data points- will often take only a few hundred milliseconds to write, depending on the database). They are also particularly useful for mirroring array values in the database, as each element will appear under a single column, and share the same data type.

Like the Standard Group, the Block Group can insert a new block, or update the first, last or a custom block. Additionally, the group can be set to only insert rows that have changed in the block.

In addition to block items, the group can have other OPC items, Tag references, and Expression items. These items can be used for triggers, handshakes, etc. They may also target a column to be written, and will write their single value to all rows in the block.

The block group is so named because it writes "blocks" of data to a database table, consisting of multiple rows and columns.

Typical Uses

Block Groups are useful in a number of situations where you need to deal with a lot of data efficiently. Mirroring/Synchronizing array values to DB - Arrays are often best stored vertically, which makes them perfect for Block Groups. Pattern mode makes configuration a breeze by allowing to you specify the array as a pattern, and set the bounds

- **Recipe management** - Like Standard Groups, but used when set points are better stored vertically than horizontally.
- **Vertical history tables** - Group data points by data type (integer, float, string), create a copy of the item that stores item path, and then use the insert changed rows option to create your own vertically storing historical tables. Create additional copies of the block item that refer to quality and timestamp in order to get further information about the data point.

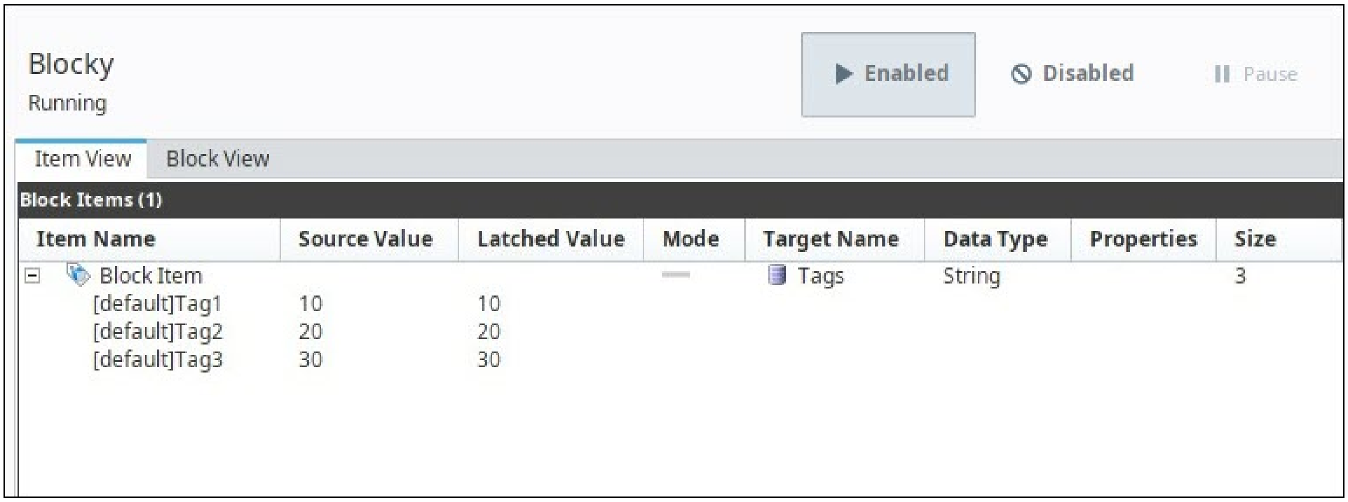
Table Format

Due to their nature, Block Groups store records in a different format than the other groups. Consider how other Transaction Groups work. A single execution of a Standard or Historical Group would store a row that looked like the following:

table_ndx	tag1	tag2	tag3

1	10	20	30

We could take the Tags from the above example, and place them in under a single block item:



Item Name	Source Value	Latched Value	Mode	Target Name	Data Type	Properties	Size
⊟ Block Item			—	Tags	String		3
[default]Tag1	10	10					
[default]Tag2	20	20					
[default]Tag3	30	30					

Under a single block item, each Tag is nested under the block item, and the block item is targeting the "Tags" column under Target name. A single execution of this group stores the records in our table as so:

table_ndx	Tags
1	10
2	20
3	30

Each additional block item would store records as a separate column.

table_ndx	Tags	More_Tags
1	10	11
2	20	22
3	30	33

Row ID and Block ID

Using the same Tag example from above, if we kept inserting new rows at every execution, our table would start to looks like the following:

table_ndx	Tags
1	10
2	20
3	30
4	15
5	25
6	35

This isn't ideal, since the table doesn't have a great way to show which value came from which Tag. To help with this, Block Groups have optional row_id and block_id columns that can be enabled (see the "Store row id" and "Store block id" settings under Group Settings). If we enable both the Block ID and Row ID, our table would look like the following:

table_ndx	Tags	row_id	block_id
1	10	0	1
2	20	1	1

3	30	2	1
4	15	0	2
5	25	1	2
6	35	2	2

Block ID represents the a single execution of the group, meaning rows with the same block_id value were inserted together. We see block_id values of 1 (colored green) are part of the same execution, and rows with a block_id value of 2 (colored blue) are a separate execution.

Row ID in an index representing which item in the block item the row corresponds to. In our example, Tag1 is the first or top item in the block item (row index 0), Tag2 is next (row index 1), and Tag3 is last (row index 2). Now we know that any value on that table with a row_id of 0 came from Tag1.

Group Settings

Beyond the differences in the data, namely that the Block Group works with multiple rows instead of just 1, this group type shares many similarities with the Standard Group.

The unique settings are:

- **Automatically create table** - If the target table does not exist, or does not have all of the required columns, it will be created/modified on group startup. If not selected and the table doesn't match, an error will be generated on startup.
- **Automatically create rows** - If the target rows do not exist, they will be created on group execution. If not selected and the rows don't match, no records will be updated.
- **Use custom index column** - If selected, you may enter any column name to hold the index. If unselected, the table index will be named <table name>_ndx.
- **Store timestamp to** - Specifies whether or not to store a timestamp with the record, and the target column. The timestamp will be generated by the group during execution. For groups that update a row(s), the timestamp will only be written if any of the values in the group are also written.
- **Store quality code to** - If selected, stores an aggregate quality for the row to the specified column. The aggregate quality is the combined quality of all of the items that write to that row. For more information about quality values, see Data Quality.
- **Store row id** - Each row will be assigned a numeric id, starting at 0. If selected, this id will also be stored with the data.
- **Store block id** - If selected, an incremental block id will be stored along with the data. This number will be 1 greater than the previous block id in the table.
- **Delete records older than** - If selected, records in the target table will be deleted after they reach the specified age. This setting is useful for preventing tables from growing in an unbounded manner, which can cause disk space and performance problems over time.

Table action

This section details how the group interacts with the table on each execution, and is not available for the Historical Group type. This means when the Timer or Schedule is active, and the Trigger condition are met. The group can insert a new row, or update the first, last or a custom record.

- **Insert New Block** - If selected, each row of the block will be inserted when the group executes, even if the data has not changed.
- **Insert changed rows** - This option will only insert the rows that have new data when the group executes. This is particularly useful for recording history for many data points on an "on change" basis, provided there is a unique id column defined. The "store row id" feature is useful for this, as well as the ability to reference the item path in an item's value property.
- **Update / Select** - This option will either update or select from matching rows based on the option selected below it. The Update Mode property above determines whether an update (OPC to DB), select (DB to OPC), or both (Bi-directional) are used when the group executes.
- **First** - Use the first row in the table. It is not recommended to use this option unless the order of the data in the table is guaranteed.
- **Last** - Use the last row in the table. This is commonly used when another group (or another program) is inserting new rows for us, and we always want to update the most recent record.
- **Custom** - Like Standard Groups, this setting allows you to target a specific section of the table, using SQL where clause syntax, with the ability to bind to dynamic item values. Unlike Standard Groups, however, the WHERE clause specified should result in enough rows to cover the block. Excess rows will not be written to, but fewer rows will result in a group warning indicating that some data could not be written.

Stored Procedure Group

The Stored Procedure Group lets you quickly map values bi-directionally to the parameters of a stored procedure. It is similar to the other groups in terms of execution, triggering, and item configuration. The primary difference is that unlike the other group types, the target is not a database table, but instead a stored procedure.

Items in the group can be mapped to input (or inout) parameters of the procedure. They also can be bound to output parameters, in which case the value returned from the procedure will be written to the item. Items can be bound to both an input and output at the same time.

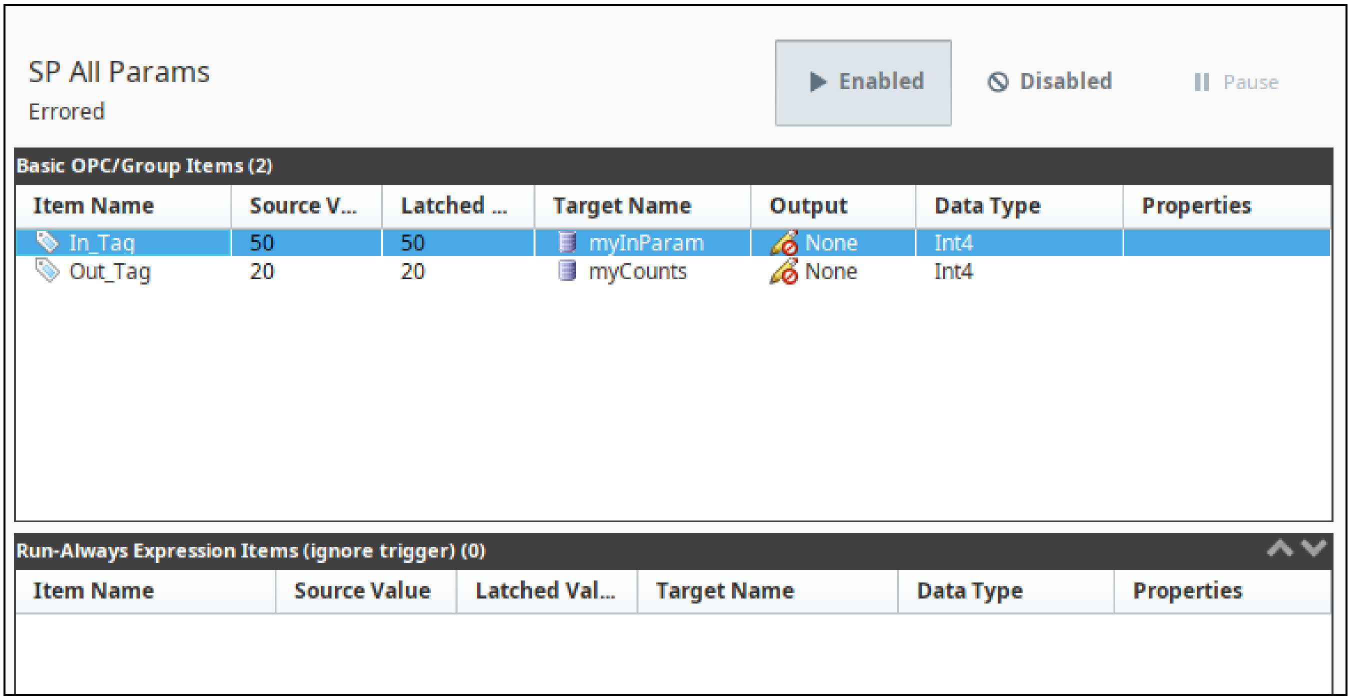
Parameters may be specified using either **parameter names** or **numerical indices**. That is, in any location where you can specify a parameter, you can either use the name defined in the database, or a 0-indexed value specifying the parameter's place in the function call.

ⓘ You cannot mix names and indices. That is, you must consistently use one or the other.

Stored Procedure Group Type

Watch the Video

If using parameter names, the names should not include any particular identifying character (for example, "?" or "@", which are used by some databases to specify a parameter).



SP All Params

Errored

Basic OPC/Group Items (2)

Item Name	Source V...	Latched ...	Target Name	Output	Data Type	Properties
In_Tag	50	50	myInParam	None	Int4	
Out_Tag	20	20	myCounts	None	Int4	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Value	Latched Val...	Target Name	Data Type	Properties

Group Settings

The Stored Procedure Group's settings look and act the same as those of the Historical Group. The primary difference, of course, is that instead of specifying a table name and column names, you'll specify a Stored Procedure and its parameters.

- **Store timestamp to** - Specifies whether or not to store a timestamp with the record, and the target column. The timestamp will be generated by the group during execution. For groups that update a row, the timestamp will only be written if any of the values in the group are also written.
- **Store quality code to** - If selected, stores an aggregate quality for the group to the specified column. The aggregate quality is the combined quality of all of the items that write to the table. For more information about quality values, see see Data Quality.
- **Procedure Name** - The name of the Stored Procedure (SP) that you will be using. You must look into the SP definition to see what inputs and outputs are available.

Typical Uses

- **Call stored procedures** - The Stored Procedure Group is the obvious choice when you want to bind values to a stored procedure. It can also be used to call procedures that take no parameters (though this can also be accomplished from Expression Items/SQLTags.
- **Replace RSSQL** - The Stored Procedure Group is very popular among users switching from RSSQL, given that application's heavy use of stored procedures.

Known Issues

When using an Oracle database, you must use indexed parameters.

Parameters in the Stored Procedure Group

When using a Stored Procedure Group, parameters may be configured to each item based on the type of the parameter:

- The **Target Name** column is used for writing, so specifying an IN or INOUT parameters under this column will have the item try to write its value to the parameter
- The **Output** column is used to move the value of an OUT or INOUT parameter into an item in the group. If an item in a group is configured to reference an OUT parameters, its **Target Name** value should be set to **Read-Only.**

Related Topics ...

- Group Update Rate

Item Types

Items are the backbone of a Transaction Group. They represent a link between a source value (derived from either an OPC value or an expression) and a cell in a database table. Items generally aren't executed in a reliable order, with the exception of **Expression** items. Expression items can be ordered using the up and down arrows located to the right of the list where the items are displayed. This can be crucial for performing complex operations that require a specific sequence. Below is a listing of each type of item.

Item Type	Description
OPC Item	Directly subscribed to an OPC server at the rate of the group. These items effectively ignore the gateway's Tag system, bypassing Tag groups and Tag providers altogether.
Expression Item	Much like an expression Tag, expression items are flexible in that their value can come from a number of different sources: specifically an expression or a database query. Expression items have two sub types: • **Run-Always** expression items are evaluated every time the group executes. Meaning, they'll run their associated expression or query every time the group executes. • **Triggered** expression items only evaluate when the group trigger is active.
Tag Reference Item	A reference to a Tag in a Tag provider. Allows a Tag to be used in a group like any other item type, except that the Tag is evaluated by its scan class instead of by the group. For more information, see the next section, Tag References vs. OPC Items. Tag Reference Items can reference the value on any Tag in a Tag provider, such as query Tags and memory Tags.

On this page ...

Item Types

Watch the Video

Tag References and OPC Items

It is easy to confuse the definition and purpose of Tag reference items and direct OPC items in Transaction Groups.

Tags may be referenced inside of Transaction Groups through a Tag Reference Item. Since the source of the Tag reference item exists outside of the Transaction Group, they have their own rules and configurations that determine when their value changes. Thus Tag reference items can have their value update according to their own execution (commonly, a Tag Group). Adding a Tag into a group is like creating a shortcut to that Tag. However, once in the group, the item can be used like any other item. Tag references are useful when it is necessary to have a single value in multiple groups, for example, as a trigger in order to coordinate execution.

OPC Items in groups (as well as expression items in groups), however, are completely executed by the group. They do not exist outside of the group in which they are defined. They are subscribed and evaluated according to the rate of the group.

Refer to the Item Type Property Table at the bottom of this page to see the properties for both Tag and OPC items.

Tag References vs. OPC Items

Watch the Video

Expression Items

Expression Items are items not driven by a PLC. Instead they allow you to configure a static value, or use some other means to automatically set a value, such as a query. They are useful for executing comparisons, simple math, and looking up values from other database tables.

Much like OPC Items, Expression Items can have alarms configured, as well as numeric scaling applied directly to the item.

Expression Items

Scope

Watch the Video

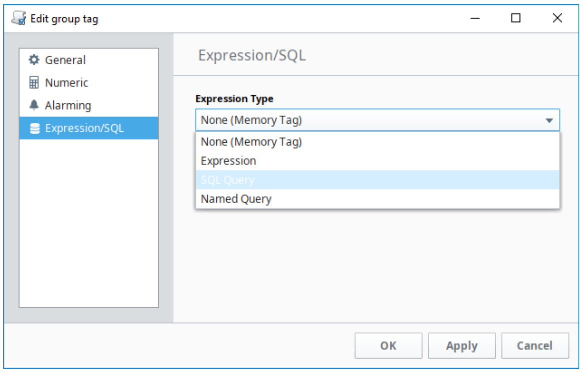
It is important to understand that an Expression Item only exists within its group, and can not be referenced by items in other Transaction Groups, Tags, and any components on a window.

Execution Order

All Expression items will evaluate in order from top to bottom. This means referencing an Expression Item above will pull the new value, but referencing an Expression Item below will give you the value from the last group execution.

Expression Type

How an Expression Item determines its value depends heavily on its type.



Expression Type: None (Memory Tag)

Expression Type	Definition
None	Behaves similar to a Memory Tag in that the value does not automatically change.
Expression	Uses Ignition's Expression Language to determine the value on the Item. The expression can reference other items in the group, as well as Tags.
SQL Query	Utilizes a SQL query to determine the item's value. Thus, a query can execute on the item and the results can be referenced by other items in the same group.
Named Query	Selecting this option will cause the value on the item to be determined by a Named Query in the same project as the Transaction Group.

Refer to the Property Table at the bottom of this page to see the Expression Item Options.

Run Always vs. Triggered Items

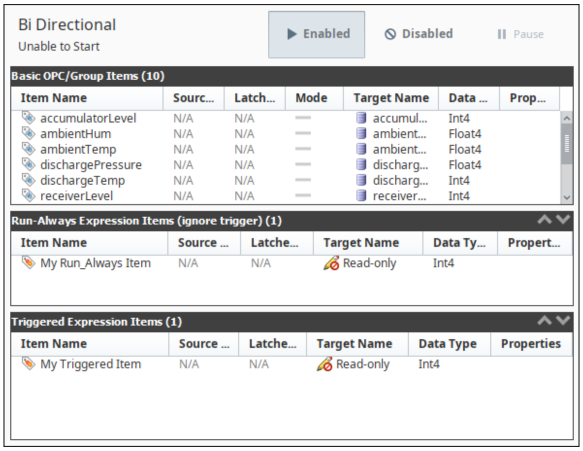
Expression Item can be configured in two different evaluation states:

- **Triggered**: The Expression Item executes only when the Transaction Group is triggered. However if the group is **not** configured to execute on a trigger, then the item will evaluate every time the group executes (similar to how the **Run-Always** state works). This is the default evaluation state new Expression Items use.
- **Run-Always**: The Expression Items will run before the group trigger is checked, so it always executes at the group's rate. This allows your expression to always evaluate regardless of the trigger in the group. Additionally, this state allows you to use the Expression Item as the trigger for the group. We advise that you never have a Target for a Run-Always Expression item

Run Always vs. Triggered Items

Watch the Video

because it always runs.



Bi Directional
Unable to Start

Basic OPC/Group Items (10)

Item Name	Sourc...	Latch...	Mode	Target Name	Data ...	Prop...
accumulatorLevel	N/A	N/A	—	accumul...	Int4	
ambientHum	N/A	N/A	—	ambient...	Float4	
ambientTemp	N/A	N/A	—	ambient...	Float4	
dischargePressure	N/A	N/A	—	discharg...	Float4	
dischargeTemp	N/A	N/A	—	discharg...	Int4	
receiverLevel	N/A	N/A	—	receiver...	Int4	

Run-Always Expression Items (ignore trigger) (1)

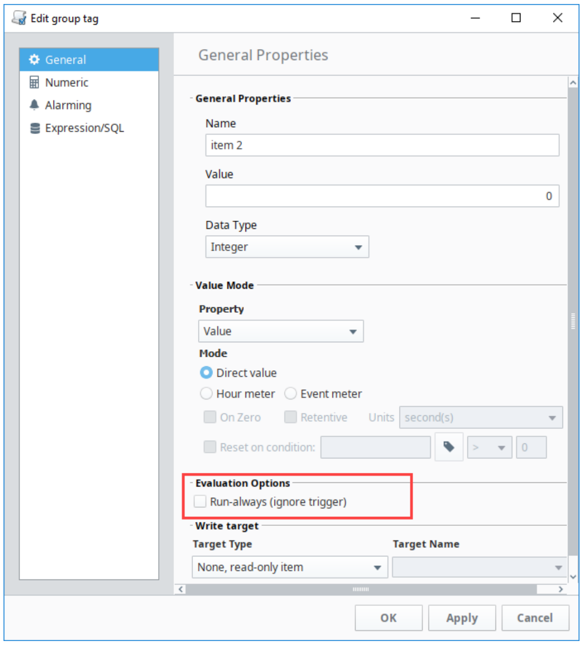
Item Name	Source ...	Latche...	Target Name	Data Ty...	Propert...
My Run_Always Item	N/A	N/A	Read-only	Int4	

Triggered Expression Items (1)

Item Name	Source ...	Latche...	Target Name	Data Type	Properties
My Triggered Item	N/A	N/A	Read-only	Int4	

Changing the Evaluation State

Toggling between the two modes can be accomplished by dragging and dropping the Expression Item to either the **Run-Always Expression Items** table or the **Triggered Expression Items** table. Alternatively, the evaluation state can be changed by editing the Expression Item and toggling the **Run-always (ignore trigger)** checkbox



Edit group tag

General Properties

General Properties

Name: item 2

Value: 0

Data Type: Integer

Value Mode

Property: Value

Mode
● Direct value
○ Hour meter ○ Event meter
☐ On Zero ☐ Retentive Units second(s)
☐ Reset on condition: > 0

Evaluation Options
☐ Run-always (ignore trigger)

Write target

Target Type: None, read-only item

Target Name:

SQL Queries and Expressions

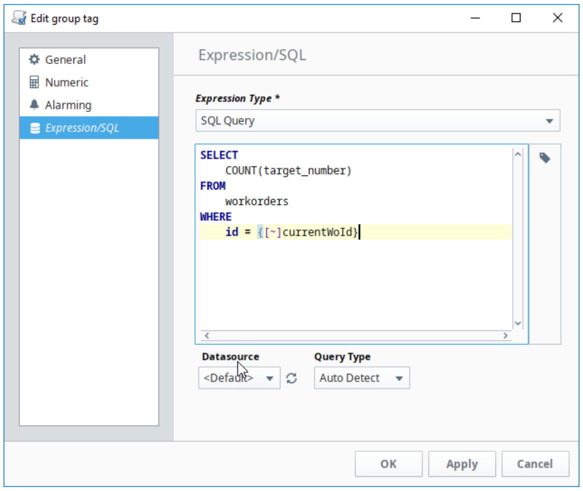
Expression items can use SQL statements and Ignition's Expression language to automatically determine the value of an Expression Item. This is useful in scenarios where you want to use a value from the database as the trigger for the Transaction Group, or aggregate several other items in the group into a single value.

Expressions and queries on an Expression Item can reference the value of other items in the group or Tags in the system by clicking on the **Tag** 🏷 icon.

There are several Expression functions available that exist only for Transaction Groups. You can find them in the **Store and Forward** and **Variables** sections of the **f(x)** function list.

SQL Query Expression Items

Watch the Video



Expression/SQL

Expression Type *: SQL Query

```
SELECT
    COUNT(target_number)
FROM
    workorders
WHERE
    id = {[~]currentWoId}
```

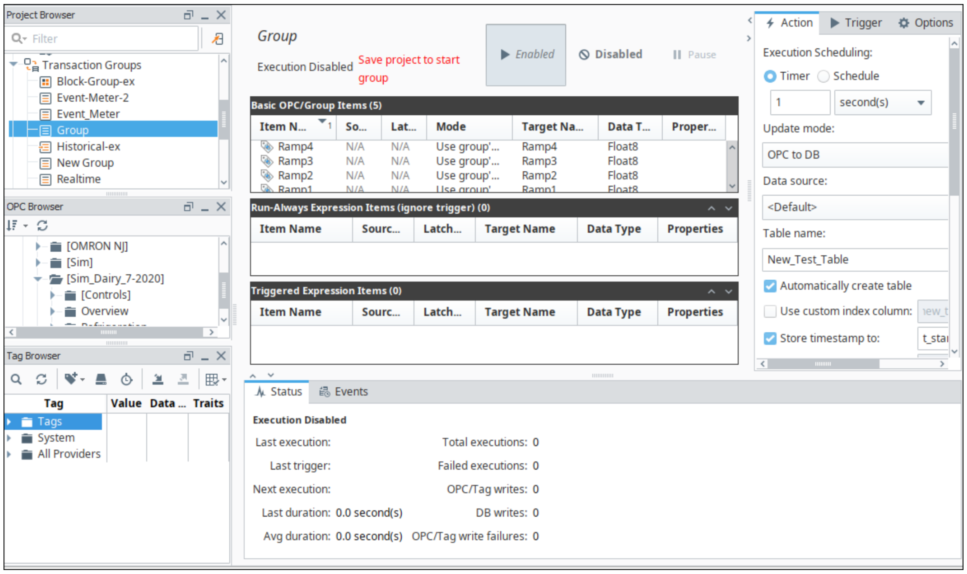
Datasource: <Default>
Query Type: Auto Detect

Creating a New Item

Below is an example of creating a new item. The steps can be applied to any item type.

1. In the Designer, go to Project Browser, and click on **Transaction Groups**.
   The workspace now changes to the Transaction Group workspace.

2. Right-click on **Transaction Group** to create a New Transaction Group, or click on a group you have previously created.
   You will now see the workspace changes to look like:



Group

Execution Disabled — Save project to start group

Basic OPC/Group Items (5)

Item N...	So...	Lat...	Mode	Target Na...	Data T...	Proper...
Ramp4	N/A	N/A	Use group'...	Ramp4	Float8	
Ramp3	N/A	N/A	Use group'...	Ramp3	Float8	
Ramp2	N/A	N/A	Use group'...	Ramp2	Float8	
Ramp1	N/A	N/A	Use group'	Ramp1	Float8	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Sourc...	Latch...	Target Name	Data Type	Properties

Triggered Expression Items (0)

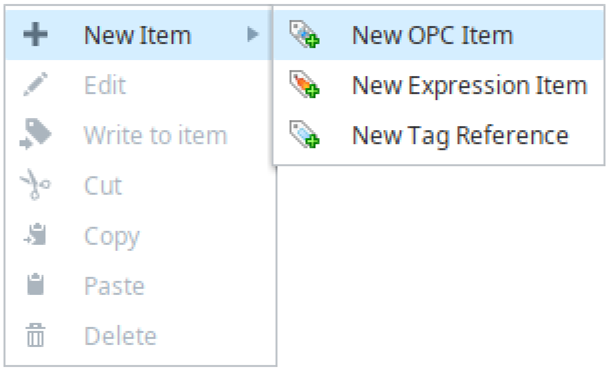
Item Name	Sourc...	Latch...	Target Name	Data Type	Properties

Execution Scheduling: ● Timer ○ Schedule
1 second(s)
Update mode: OPC to DB
Data source: <Default>
Table name: New_Test_Table
☑ Automatically create table
□ Use custom index column: new_t
☑ Store timestamp to: t_stai

Execution Disabled

Last execution:		Total executions:	0
Last trigger:		Failed executions:	0
Next execution:		OPC/Tag writes:	0
Last duration:	0.0 second(s)	DB writes:	0
Avg duration:	0.0 second(s)	OPC/Tag write failures:	0

3. Right-click in the white area, and choose **New Item > New OPC Item**. The options in the popups represent the different item types. Refer to the Item Type Property Table on this page for more information on the various item types and their properties.



4. Once you configured the item, click **OK**.

Item Type Property Table

The following tables describes the OPC, Tag and Expression Item properties.

OPC Item Options

Property	Description
General	
Name	The name of the OPC item in the group. There cannot be duplicate names within a group.
Data Type	The data type used to read values from the PLC.
OPC Properties	
OPC Server	The Selected OPC Server. This is a dropdown list showing all the OPC Servers added in the Ignition Gateway.
OPC Item Path	The OPC address assigned by the server. Dragging and dropping from the OPC Browser will automatically populate this field. The following feature is new in Ignition version **8.1.5** Click here to check out the other new features As of 8.1.5 it's possible to escape curly braces {} in the item path by using additional curly braces. For example: `{{device_name}}` would evaluate to `{<device_name param value>}.`
Source Data Type	Data type for the OPC item.
Value Mode	
Property	Which property of the OPC item you want to use. • Value - Item value • Quality - Quality code from OPC Server (192 = GOOD_DATA) • Timestamp - The last time the item value changed • Name - The SQLBridge Item Name property of this Item **Note:** UDT Instances used in Transaction groups only support String datatypes. Other datatypes could cause Transaction Groups groups running in previous versions of Ignition to fault.
Mode	Options for displaying values based on the Item value. • Direct Value - Item value • Hour Meter - Record the amount of time the Item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The data type should be set to integer or float when using an Hour Meter regardless of the OPC Item type. ○ On Zero - Use a zero value to accumulate time instead of a non-zero value ○ Retentive - Retain the Hour Meter value when it is not accumulating. ○ Units - The time units to display. • Event Meter - Record the number or times the Item value is non-zero. The data type should be set to integer when using an Event Meter regardless of the OPC Item type. ○ On Zero - Use a zero value to accumulate events instead of a non-zero value
Write Target	
Mode	Changes the items directional read/write option. • Use group's mode - Inherit the Update Mode from the Item's Group. • OPC to DB - Only read from the OPC server and write to the database. • DB to OPC - Only read from the database and write to the OPC Server. • Bi-directional OPC wins - Read and Write to both the database and OPC Server. On group start, write OPC Server values to the database.

	• Bi-directional DB wins - Read and Write to both the database and OPC Server. On group start, write database values to the OPC Server.
Target Type	This is the selection for what the Item will write to when the group executes. • None, read-only item - Do not write this value to the database. • Database field - Write the Item value to the specified column in the database table. This list will populate with all the column names from the Group's target table after the first time the group is run.
Target Name	The name of the column in the database that this Item will write to when the group executes. The Target Name list will populate with all the column names from the Group's target table if the Target Type is Database field.
Alarming	The Alarming settings for the OPC items. See Alarming Properties for a full explanation.

Tag Reference Item Options

General	
Name	The name of the OPC item in the group. There cannot be duplicate names within a group.
Tag Path	The path to the tag being referenced. This value is not editable except by clicking the Insert Tag button. There cannot be duplicate names within a group.
Data Type	The data type to write to into the database if this item is not read-only.
Value Mode	
Property	Which property of the Tag you want to use. • Value - Item value • Quality - Quality code of the Tag (192 = GOOD_DATA) • Timestamp - The last time the item value changed • Name - The SQLBridge Item Name property of this Item.
Mode	Options for displaying values based on the Item value. • Direct Value - Item value • Hour Meter - Record the amount of time the Item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The data type should be set to integer or float when using an Hour Meter regardless of the OPC Item type. On Zero - Use a zero value to accumulate time instead of a non-zero value Retentive - Retain the Hour Meter value when it is not accumulating. Units - The time units to display. • Event Meter - Record the number or times the Item value is non-zero. The data type should be set to integer when using an Event Meter regardless of the OPC Item type. On Zero - Use a zero value to accumulate events instead of a non-zero value
Write Target	
Mode	Changes the items directional read/write option. This is only editable when the target Type is set to Database field. • Use group's mode - Inherit the Update Mode from the Item's Group. • OPC to DB - Only read from the OPC server and write to the database. • DB to OPC - Only read from the database and write to the OPC Server. • Bi-directional OPC wins - Read and Write to both the database and OPC Server. On group start, write OPC Server values to the database. • Bi-directional DB wins - Read and Write to both the database and OPC Server. On group start, write database values to the database.
Target Type	This is the selection for what the Item will write to when the group executes. • None, read-only item - Do not write this value to the database. • Database field - Write the Item value to the specified column in the database table.
Target Name	The name of the column in the database that this Item will write to when the group executes. The Target Name list will populate with all the column names from the Group's target table if the Target Type is Database field.

Expression Item Options

General	
Name	The name of the OPC item in the group. There cannot be duplicate names within a group.
Value	The static value of this Expression item. This will be overwritten by an Expression/SQL binding.
Data Type	The data type values are stored as.
Value Mode	
Property	Which property of the OPC item you want to use. • Value - Item value • Quality - Quality code of the expression/SQL Query (192 = GOOD_DATA) • Timestamp - The last time the item value changed. • Name - The SQLBridge Item Name property of this Item.
Mode	Options for displaying values based on the Item value. • Direct Value - Item value • Hour Meter - Record the amount of time the Item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The data type should be set to integer or float when using an Hour Meter regardless of the OPC Item type. On Zero - Use a zero value to accumulate time instead of a non-zero value Retentive - Retain the Hour Meter value when it is not accumulating. Units - The time units to display. • Event Meter - Record the number or times the Item value is non-zero. The data type should be set to integer when using an Event Meter regardless of the OPC Item type. On Zero - Use a zero value to accumulate events instead of a non-zero value
Evaluation Mode	Run-always (ignore Trigger) - When selected, this causes the group to evaluate at each group interval, before the trigger state is evaluated.
Write Target	Target type - This is the selection for what the Item will write to when the group executes. • None, read-only item - Do not write this value to the database. • Database field - Write the Item value to the specified column in the database table. • Other Tag - Write the Expression Item's value back to an OPC item or Tag Reference.
Target Name	The name of the column in the database that this Item will write to when the group executes. The Target Name list will populate with all the OPC Item and Tag Reference names from this Group, or the column names from the Group's target table depending on the Target Type selected.
Numeric	These are the Numeric properties for Expression Items. For a full description, see Tag Scaling Properties.
Alarming	These are the Alarming settings for the OPC items. See Alarming Properties for a full explanation.
Expression	These are the Expression/SQL Query options for Expression Items. See Expression/SQL Properties for a full explanation.

Hour and Event Meters

Hour meter and **Event meter** refer to the **Value Mode** option settings on Tags in Transaction Groups. The Value mode drives values that are used to create values that determine how long a value was true. While the selected Value Mode for most transactions is **Direct**, however, the **Hour meter** mode accumulates value for the duration of a condition, and the **Event meter** accumulates count in response to the condition.

INDUCTIVE UNIVERSITY

Hour and Event Meters

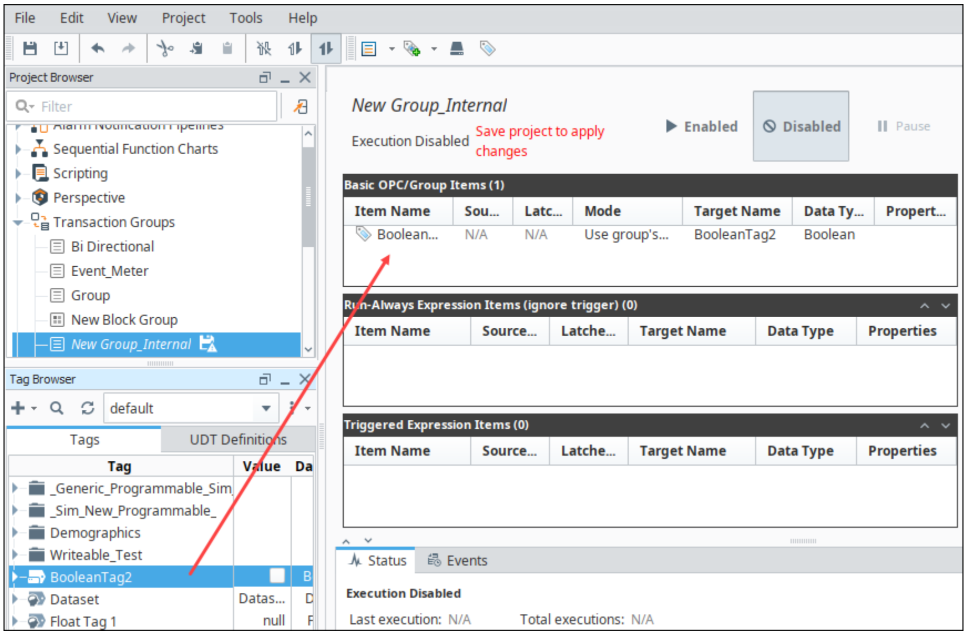
Watch the Video

Hour Meter

It is common to write to a Tag during the time when a Tag's value is true. An hour meter simplifies this effort. Hour meters can be meters that accumulate the millisecond, second, minute, hour, or day.

Count the Duration of a Tag Being True

1. From the Tag Browser, drag a boolean Tag into the **Basic OPC/Groups Items** area of a **Standard** Transaction Group.



New Group_Internal

Execution Disabled Save project to apply changes

Basic OPC/Group Items (1)

Item Name	Sou...	Latc...	Mode	Target Name	Data Ty...	Propert...
Boolean...	N/A	N/A	Use group's...	BooleanTag2	Boolean	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source...	Latche...	Target Name	Data Type	Properties

Triggered Expression Items (0)

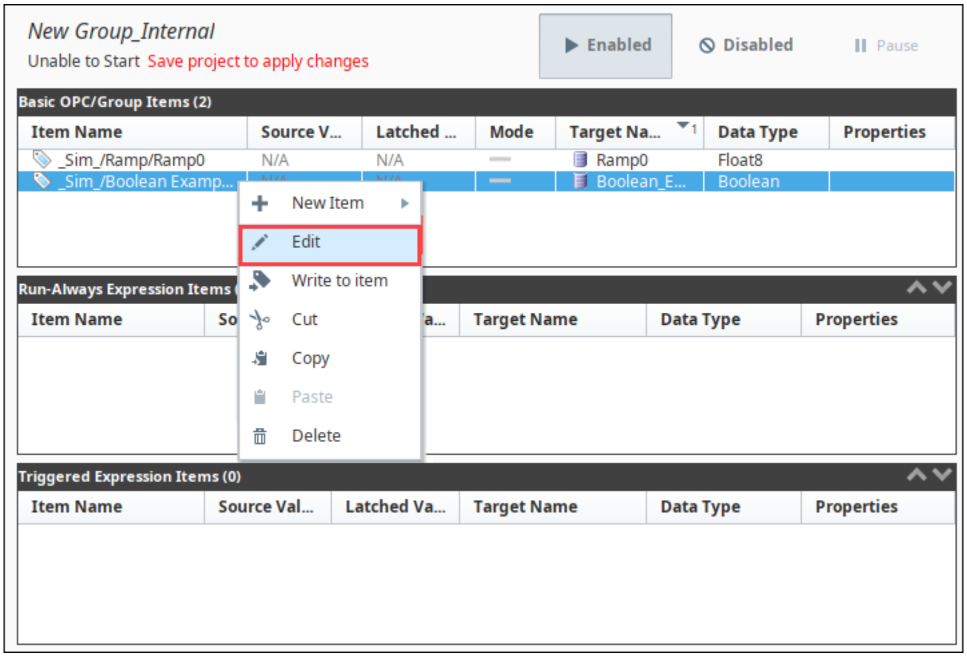
Item Name	Source...	Latche...	Target Name	Data Type	Properties

Execution Disabled

Last execution: N/A Total executions: N/A

2. From OPC Browser or Tag Browser, drag a memory tag or OPC tag (must be a numeric data type) into the **Basic OPC/Groups Items** portion of the **Standard** Transaction Group.

3. Right-click on the boolean tag in the Transaction Group and select **Edit** to edit it.
   The **Edit group tag** window is displayed.



New Group_Internal

Unable to Start Save project to apply changes

Basic OPC/Group Items (2)

Item Name	Source V...	Latched ...	Mode	Target Na...	Data Type	Properties
Sim/Ramp/Ramp0	N/A	N/A	—	Ramp0	Float8	
Sim/Boolean Examp...	N/A	N/A	—	Boolean_E...	Boolean	

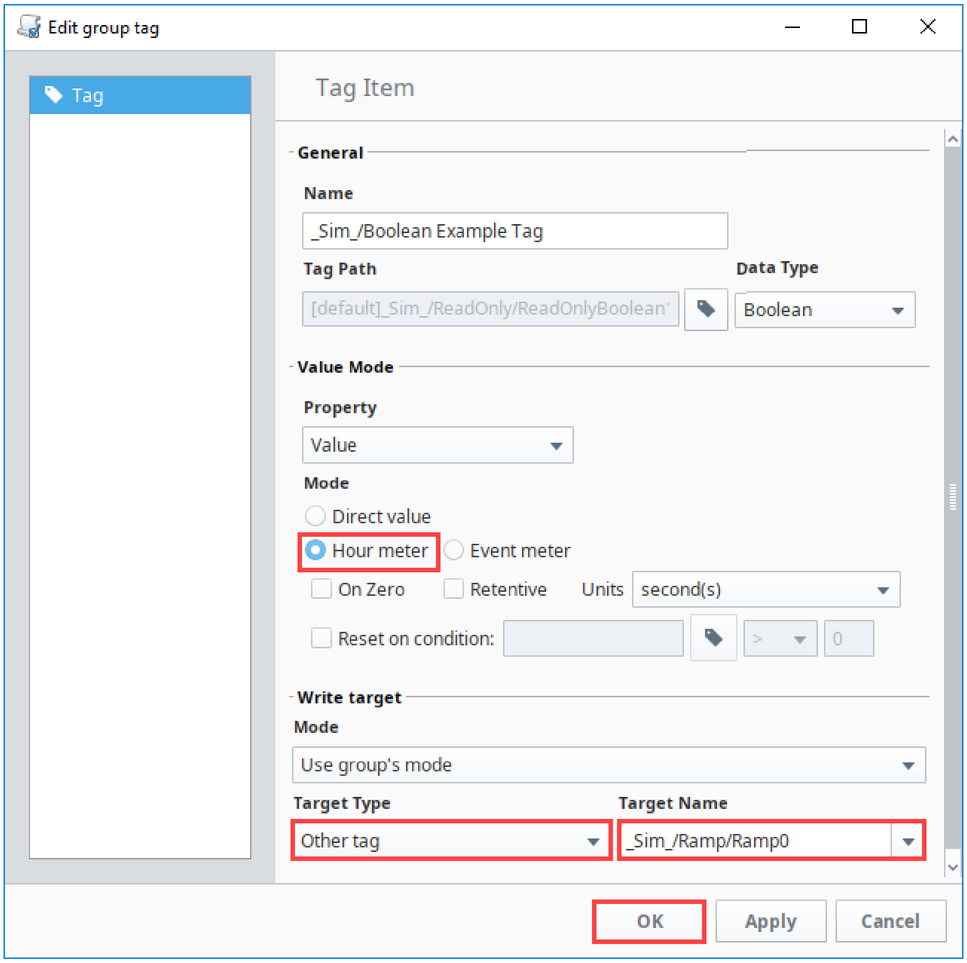
Run-Always Expression Items

Item Name	So...	...a...	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source Val...	Latched Va...	Target Name	Data Type	Properties

4. In the **Edit group tag** window, for **Value Mode**, select **Hour meter**.
5. In the **Edit group tag** window, for **Target Type** select **Other tag** from the dropdown menu, and in **Target Name** enter the name of the memory tag as the target, and click **OK**.



Edit group tag

Tag Item

General

Name: _Sim_/Boolean Example Tag

Tag Path: [default]_Sim_/ReadOnly/ReadOnlyBoolean'

Data Type: Boolean

Value Mode

Property: Value

Mode

○ Direct value

● Hour meter ○ Event meter

□ On Zero □ Retentive Units second(s)

□ Reset on condition: > 0

Write target

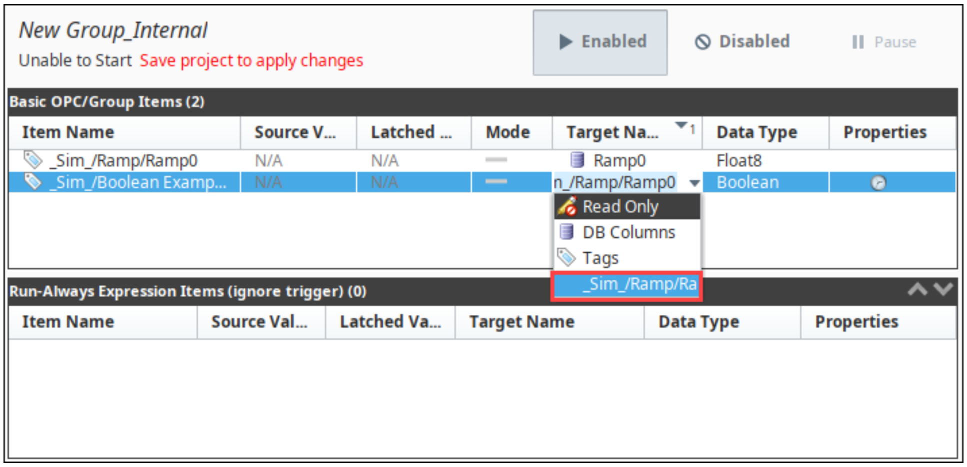
Mode: Use group's mode

Target Type: Other tag

Target Name: _Sim_/Ramp/Ramp0

6. In the **Basic OPC/Groups Items** area where the tags are located, go to the **Target Name** column, left-click on each tag to get the dropdown menu, and set the following:

a. For the boolean tag, select the memory tag from the dropdown which is set previously as the target tag to write the hour meter to.



New Group_Internal

Unable to Start Save project to apply changes

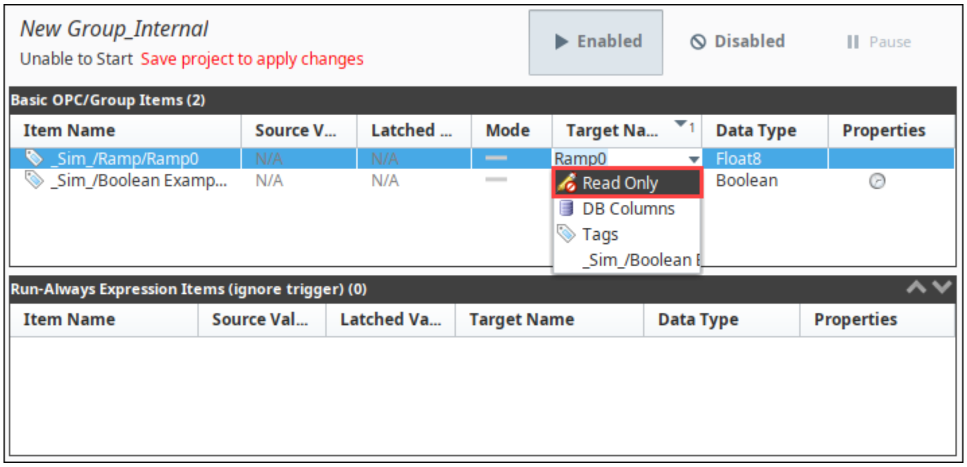
Basic OPC/Group Items (2)

Item Name	Source V...	Latched ...	Mode	Target Na...	Data Type	Properties
Sim/Ramp/Ramp0	N/A	N/A	—	Ramp0	Float8	
Sim/Boolean Examp...	N/A	N/A	—	n_/Ramp/Ramp0	Boolean	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Val...	Latched Va...	Target Name	Data Type	Properties

b. For the memory Tag, select Read-only from the dropdown.



New Group_Internal

Unable to Start Save project to apply changes

Basic OPC/Group Items (2)

Item Name	Source V...	Latched ...	Mode	Target Na...	Data Type	Properties
Sim/Ramp/Ramp0	N/A	N/A	—	Ramp0	Float8	
Sim/Boolean Examp...	N/A	N/A	—		Boolean	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Val...	Latched Va...	Target Name	Data Type	Properties

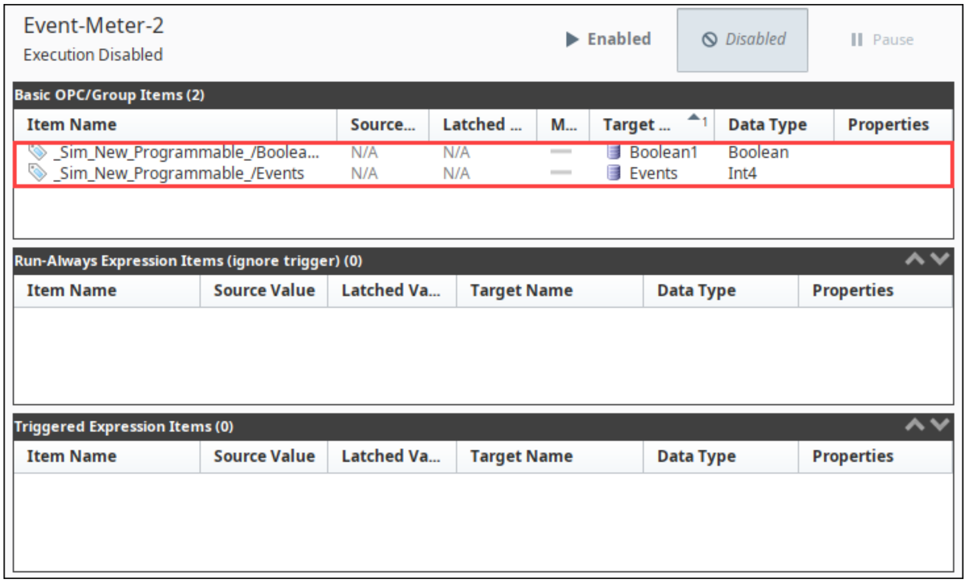
7. Click **Enabled** at the top of the page, and do a **File > Save** to start the group.
8. Make the boolean Tag true to the start the Hour meter.

Event Meter

Another common scenario is to count the number of times an event occurred. For example, where there is boolean Tag and you want to count the number of cycles the boolean Tag has experienced.

Count in Response to a Tag being True

1. From OPC Browser or Tag Browser, drag a boolean Tag into the **Basic OPC/Groups Items** area of a **Standard** Transaction Group.
2. From OPC Browser or Tag Browser, drag a memory Tag or OPC Tag (must be a numeric data type) into the **Basic OPC/Groups Items** portion of the **Standard** Transaction Group.



Event-Meter-2

Execution Disabled

Basic OPC/Group Items (2)

Item Name	Source...	Latched ...	M...	Target ...	Data Type	Properties
_Sim_New_Programmable_/Boolea...	N/A	N/A	—	Boolean1	Boolean	
_Sim_New_Programmable_/Events	N/A	N/A	—	Events	Int4	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Value	Latched Va...	Target Name	Data Type	Properties

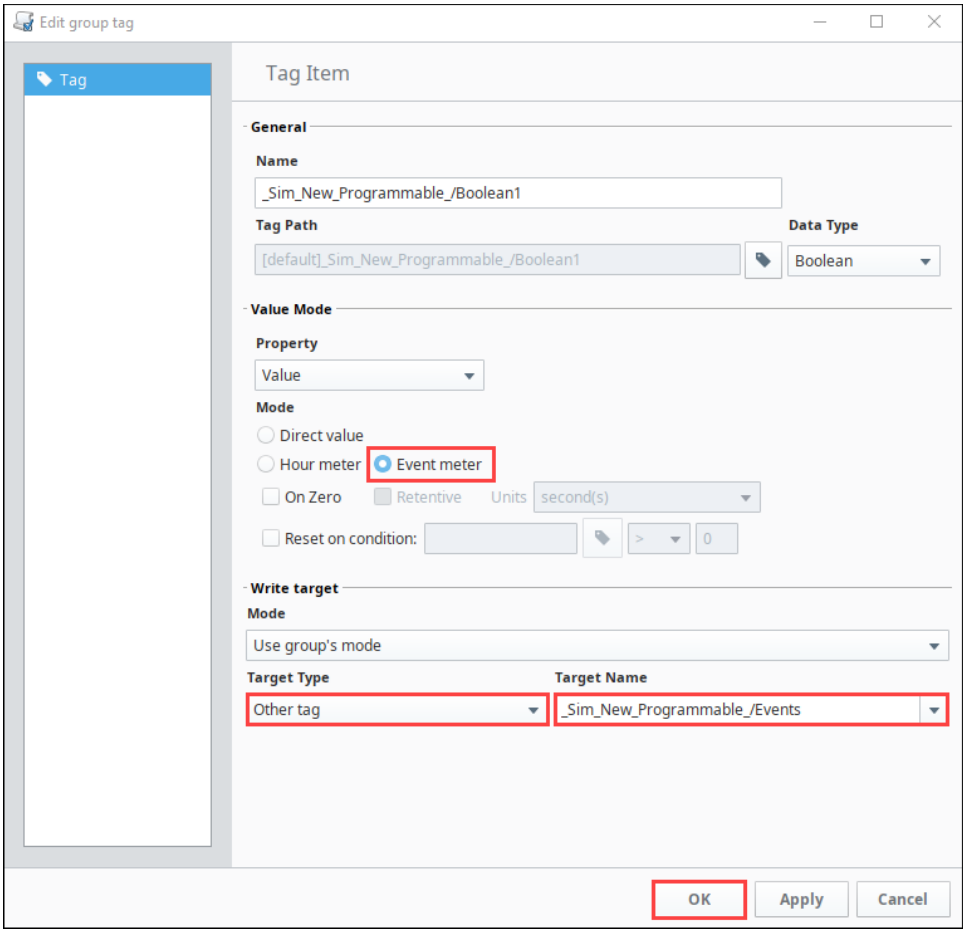
Triggered Expression Items (0)

Item Name	Source Value	Latched Va...	Target Name	Data Type	Properties

3. Right-click on the boolean Tag in the Transaction Group and select **Edit** to edit it.
4. In the **Edit group tag** window, set the following:

   Value Mode: **Event meter**
   Target Type: **Other Tag**
   Target Name: **_Sim_New_Programmable_/Events** (or name of the Tag you are using)

5. Click **OK**.



Edit group tag

Tag Item

General

Name: _Sim_New_Programmable_/Boolean1

Tag Path: [default]_Sim_New_Programmable_/Boolean1

Data Type: Boolean

Value Mode

Property: Value

Mode

○ Direct value
○ Hour meter ● Event meter
□ On Zero □ Retentive Units second(s)
□ Reset on condition: > 0

Write target

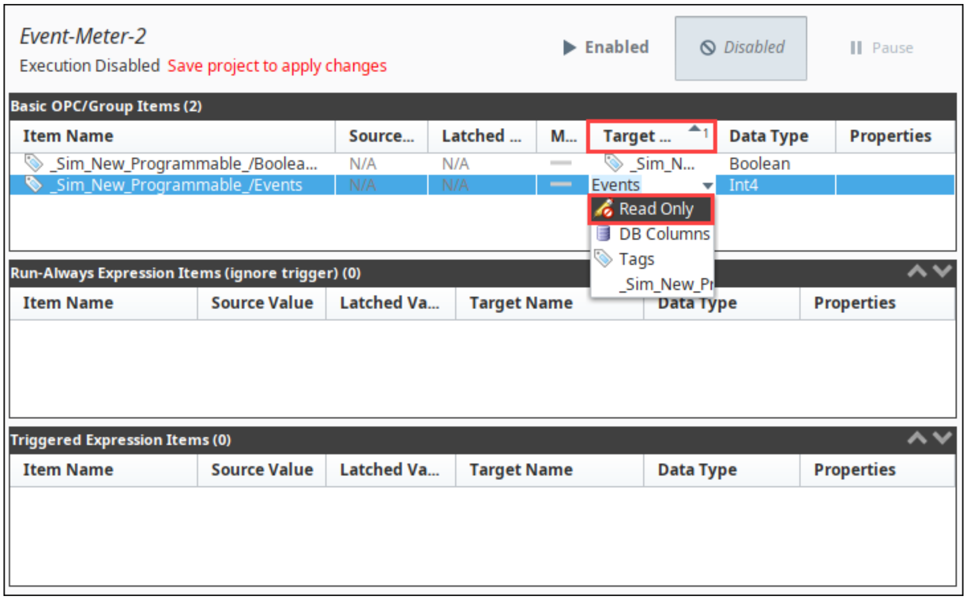
Mode: Use group's mode

Target Type: Other tag

Target Name: _Sim_New_Programmable_/Events

6. In the **Basic OPC/Groups Items** area where the Tags are located, go to the **Target Name** column. Left-click on each Tag to get the dropdown menu, and set the following:

a. For the boolean Tag, select the memory Tag from the dropdown which is set previously as the target Tag to write the hour meter to.
b. For the memory Tag, select Read-only from the dropdown.


Event-Meter-2

Execution Disabled Save project to apply changes

Basic OPC/Group Items (2)

Item Name	Source...	Latched ...	M...	Target ...	Data Type	Properties
_Sim_New_Programmable_/Boolea...	N/A	N/A	—	_Sim_N...	Boolean	
_Sim_New_Programmable_/Events	N/A	N/A	—	Events	Int4	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Value	Latched Va...	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source Value	Latched Va...	Target Name	Data Type	Properties

7. Click **Enabled** at the top of the page, and do a **File > Save All** to start the group.
8. Make the boolean Tag true to start the Event meter. You'll see the Event Tag update in the Tag Browser.

Reset an Hour or Event Meter Based on a Condition

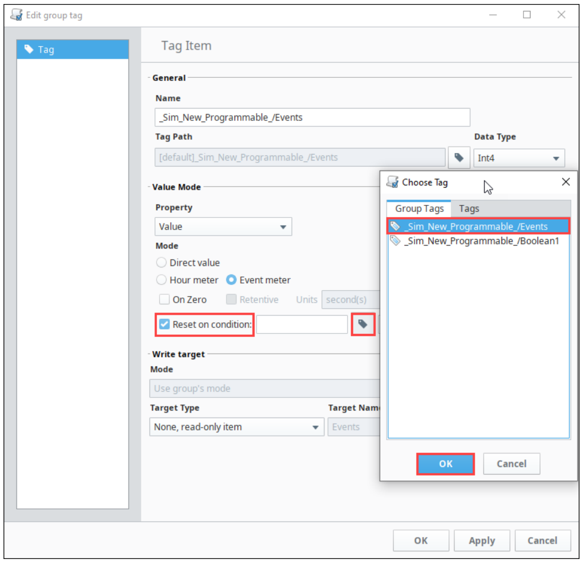
INDUCTIVE UNIVERSITY

Resetting Hour and Event Meters

Watch the Video

You can set the hour or event meter based on a condition.

1. In the Basic OPC/Group Items section, right-click and Edit a Tag that is serving as the Hour or Event meter.
   The Edit group tag window is displayed.
2. In the **Value Mode** area, select the **Reset on condition** check box.
3. Click the **Tag** icon to display the **Choose Tag** window, and select a Group Tag from the popup window.
4. Click **OK**.


Edit group tag

Tag Item

General

Name: _Sim_New_Programmable_/Events

Tag Path: [default]_Sim_New_Programmable_/Events

Data Type: Int4

Value Mode

Property: Value

Mode: ○ Direct value ○ Hour meter ● Event meter

□ On Zero □ Retentive Units second(s)

☑ Reset on condition:

Write target

Mode: Use group's mode

Target Type: None, read-only item

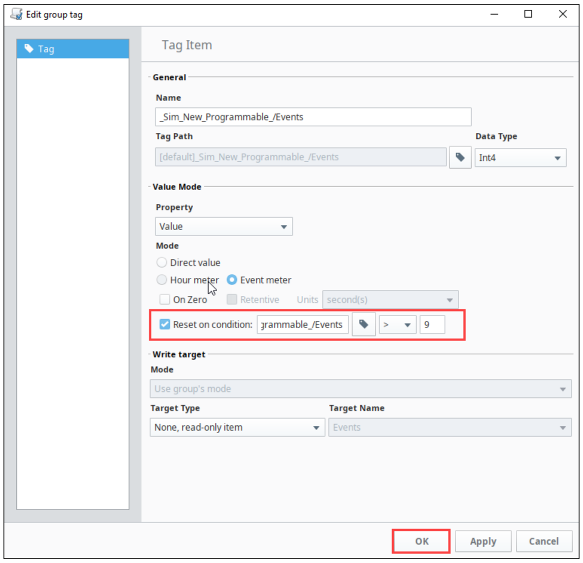
Target Name: Events

Choose Tag

Group Tags | Tags

_Sim_New_Programmable_/Events
_Sim_New_Programmable_/Boolean1

5. Next to the **Tag** icon, choose the **operator** sign (for example **>**), and enter a number. In the example we entered **9**.



General

Name: _Sim_New_Programmable_/Events

Tag Path: [default]_Sim_New_Programmable_/Events

Data Type: Int4

Value Mode

Property: Value

Mode

○ Direct value

○ Hour meter ● Event meter

□ On Zero □ Retentive Units second(s)

☑ Reset on condition: ʒrammable_/Events > 9

Write target

Mode: Use group's mode

Target Type: None, read-only item

Target Name: Events

6. Click **OK**. The target Tag will now reset in response to the condition (after nine occurrences in our example).

Next...

- Trigger Options

Transaction Group Examples

Transaction Groups

There are four basic types of Transaction Groups that can be used in Ignition:

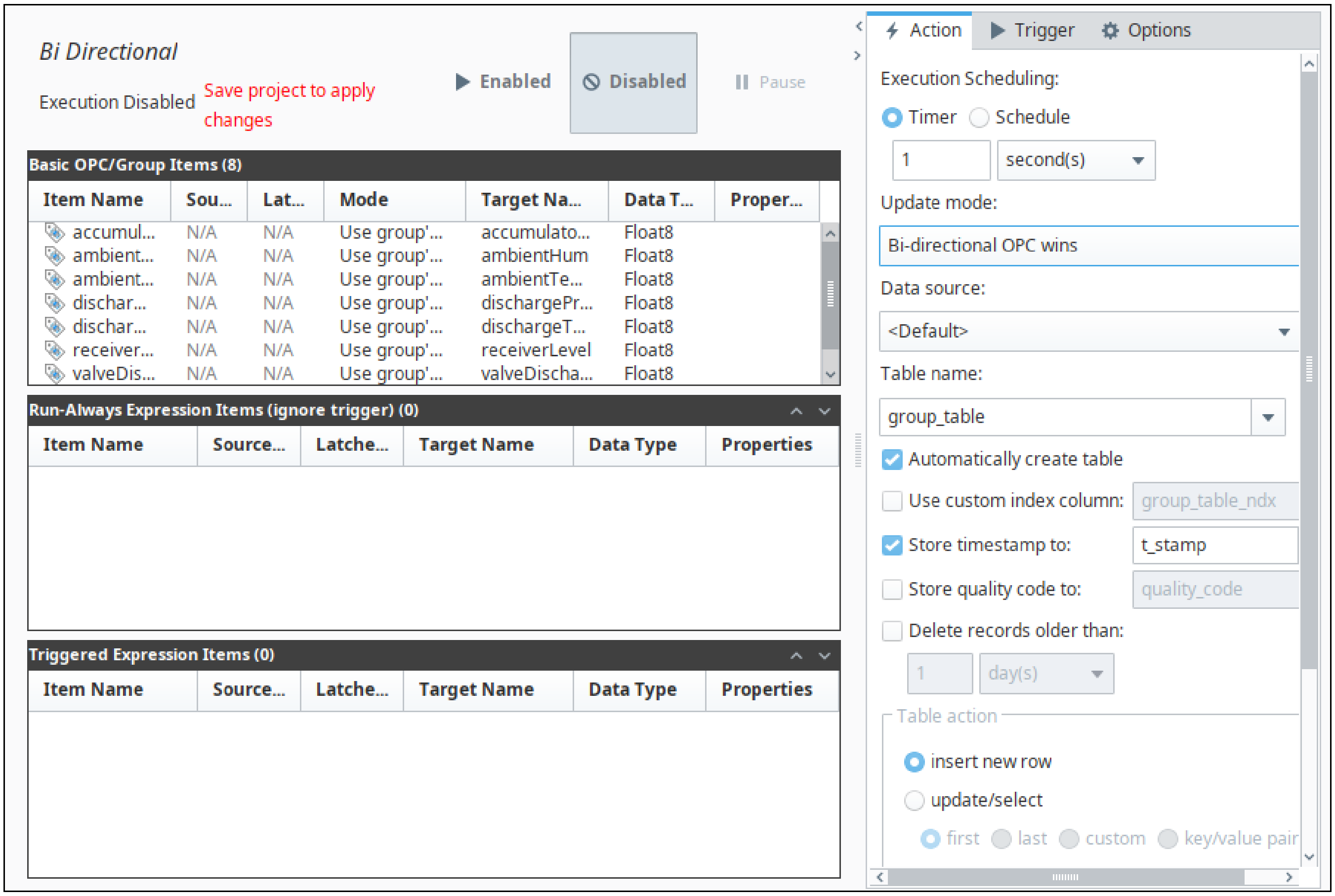
- **Standard**: The heart of bi-directional data storage and management
- **Historical**: Simple historical trending
- **Block**: Efficient large scale data storage
- **Stored Procedure**: Interact with existing protected data systems

This Section has examples for each type of group and shows the different ways that you can use them. For a more complete understanding of how the parts of each group works, see Understanding Transaction Groups.

On this page ...

Standard Group

The Standard Group is the most flexible group. It is commonly used as a bi-directional sync between your PLCs and databases. In addition to this, it can also be used to push data in either direction. This means the Standard Group can be used to store historical data, add to/update existing tables, and create recipe management tools.



Bi Directional

Execution Disabled Save project to apply changes

Basic OPC/Group Items (8)

Item Name	Sou...	Lat...	Mode	Target Na...	Data T...	Proper...
accumul...	N/A	N/A	Use group'...	accumulato...	Float8	
ambient...	N/A	N/A	Use group'...	ambientHum	Float8	
ambient...	N/A	N/A	Use group'...	ambientTe...	Float8	
dischar...	N/A	N/A	Use group'...	dischargePr...	Float8	
dischar...	N/A	N/A	Use group'...	dischargeT...	Float8	
receiver...	N/A	N/A	Use group'...	receiverLevel	Float8	
valveDis...	N/A	N/A	Use group'...	valveDischa...	Float8	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source...	Latche...	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source...	Latche...	Target Name	Data Type	Properties

Execution Scheduling:

● Timer ○ Schedule

1 second(s)

Update mode:

Bi-directional OPC wins

Data source:

<Default>

Table name:

group_table

☑ Automatically create table

□ Use custom index column: group_table_ndx

☑ Store timestamp to: t_stamp

□ Store quality code to: quality_code

□ Delete records older than:

1 day(s)

Table action

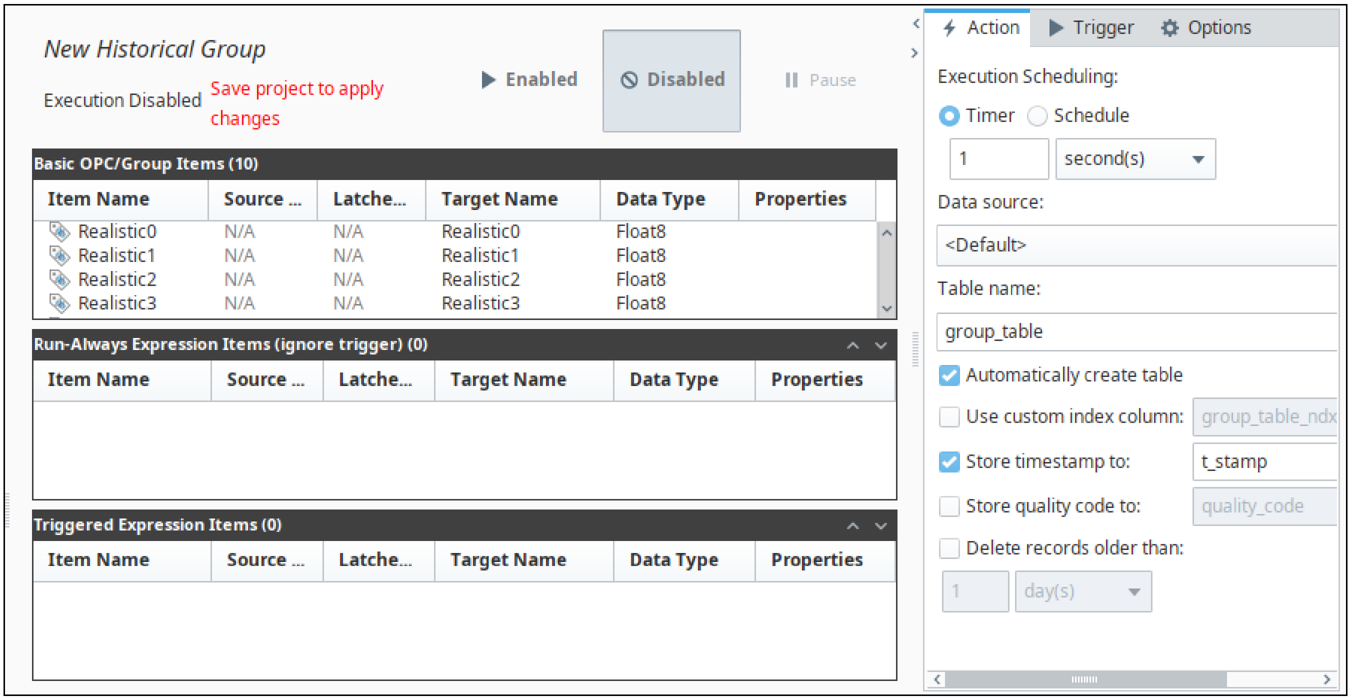
● insert new row

○ update/select

● first ○ last ○ custom ○ key/value pair

Historical Group

The Historical Group is the most straightforward and simplest to use. It will take OPC data and store it as history in a database.



New Historical Group

Execution Disabled Save project to apply changes

Basic OPC/Group Items (10)

Item Name	Source ...	Latche...	Target Name	Data Type	Properties
Realistic0	N/A	N/A	Realistic0	Float8	
Realistic1	N/A	N/A	Realistic1	Float8	
Realistic2	N/A	N/A	Realistic2	Float8	
Realistic3	N/A	N/A	Realistic3	Float8	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source ...	Latche...	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source ...	Latche...	Target Name	Data Type	Properties

Execution Scheduling:

● Timer ○ Schedule

1 second(s)

Data source:

<Default>

Table name:

group_table

☑ Automatically create table

□ Use custom index column: group_table_ndx

☑ Store timestamp to: t_stamp

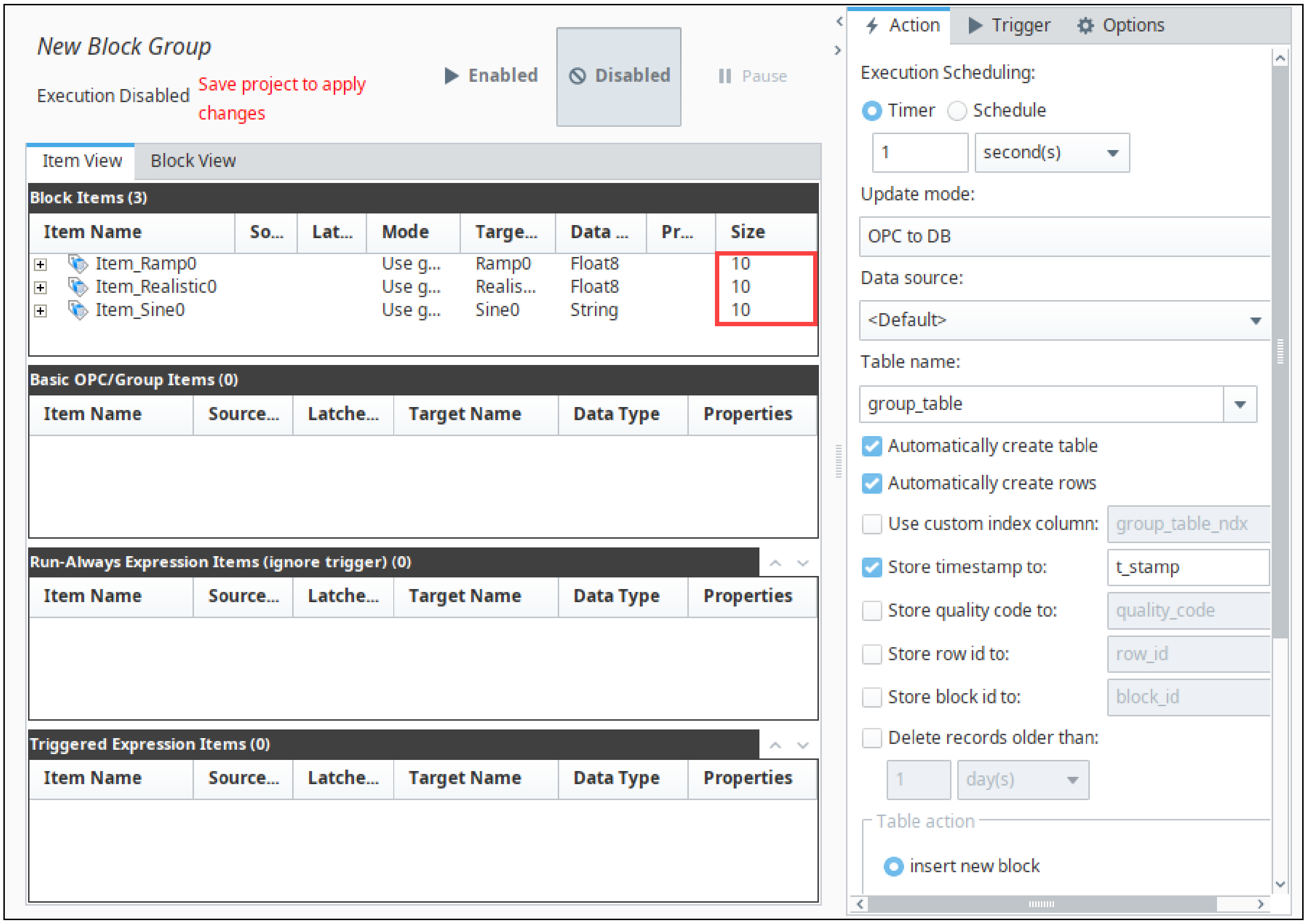
□ Store quality code to: quality_code

□ Delete records older than:

1 day(s)

Block Group

The Block Group is used to efficiently store large amounts of data in blocks or chunks of similar data in the database. This is very useful if you have many devices with the same Tags in them.



New Block Group

Execution Disabled Save project to apply changes

Block Items (3)

Item Name	So...	Lat...	Mode	Targe...	Data ...	Pr...	Size
Item_Ramp0			Use g...	Ramp0	Float8		10
Item_Realistic0			Use g...	Realis...	Float8		10
Item_Sine0			Use g...	Sine0	String		10

Basic OPC/Group Items (0)

Item Name	Source...	Latche...	Target Name	Data Type	Properties

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source...	Latche...	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source...	Latche...	Target Name	Data Type	Properties

Execution Scheduling:

● Timer ○ Schedule

1 second(s)

Update mode:

OPC to DB

Data source:

<Default>

Table name:

group_table

☑ Automatically create table

☑ Automatically create rows

□ Use custom index column: group_table_ndx

☑ Store timestamp to: t_stamp

□ Store quality code to: quality_code

□ Store row id to: row_id

□ Store block id to: block_id

□ Delete records older than:

1 day(s)

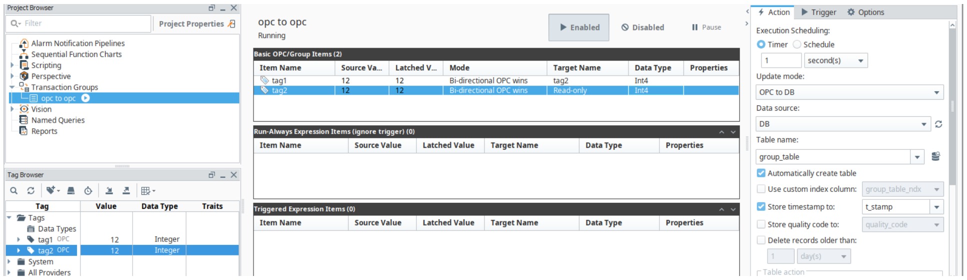
Table action

● insert new block

Configuring Transaction Group for OPC to OPC Interaction

Transaction Groups are generally used to channel OPC data to a database or vice-versa. It is also possible to configure your Standard Transaction Group to be able to get information from one OPC data point to another. This is useful in the event that you have tags coming from one PLC and you need the tag information to be sent to another PLC on your plant floor.

1. Create a Standard Transaction Group and from your Tag browser.
2. Drag two Tags into your Transaction Group's Basic OPC/Group Items section. For this example, the Tags will be called tag1 and tag2 and they will be coming from two different PLCs.
3. Set the mode on tag1 to **Bi-directional OPC Wins** and set its Target Name to be **tag2**.
4. Set the mode on tag2 to be **Bi-directional OPC Wins** and set its Target Name to **Read-only**. The mode on tag2 is not as important here as it is a read-only item, but we set it to Bi-directional OPC Wins anyway.



Tag	Value	Data Type	Traits
Tags			
Data Types			
tag1 OPC	12	Integer	
tag2 OPC	12	Integer	
System			
All Providers			

opc to opc
Running

Basic OPC/Group Items (2)

Item Name	Source Va...	Latched V...	Mode	Target Name	Data Type	Properties
tag1	12	12	Bi-directional OPC wins	tag2	Int4	
tag2	12	12	Bi-directional OPC wins	Read-only	Int4	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties

Execution Scheduling:
● Timer ○ Schedule
1 second(s)
Update mode: OPC to DB
Data source: DB
Table name: group_table
☑ Automatically create table
☐ Use custom index column: group_table_ndx
☑ Store timestamp to: t_stamp
☐ Store quality code to: quality_code
☐ Delete records older than: 1 day(s)
Table action

This will make sure that every 1 second, the value from tag1 will be written to tag2.

Related Topics ...

- Understanding Transaction Groups

In This Section ...

Block Group

The Block group is a type of Transaction Group that stores data vertically. Whereas, a Standard group stores the information horizontally in a single row. Block groups share many of the same features as the Standard group. They can be bidirectional, insert into a database, or simply update the database. All the rows in a Block group are associated with a single database transaction therefore the process of writing to the database is very efficient.

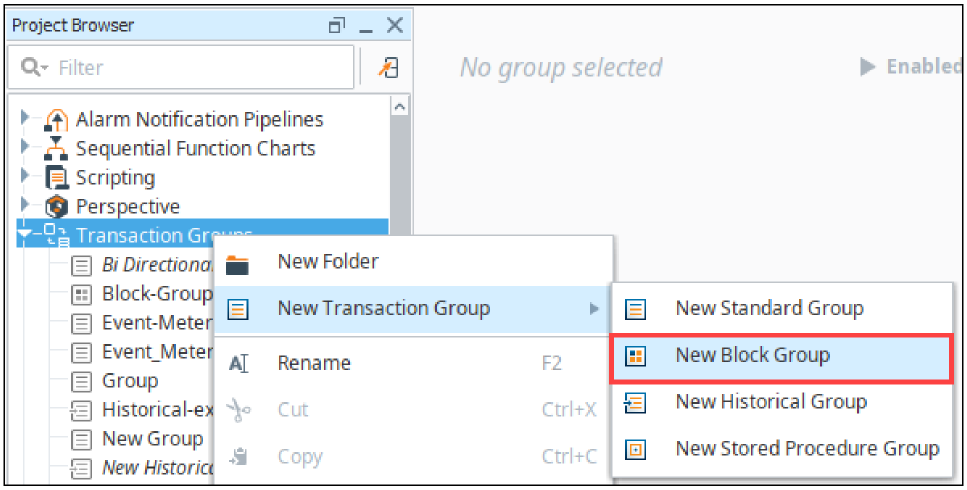
On this page ...

Block Group Type

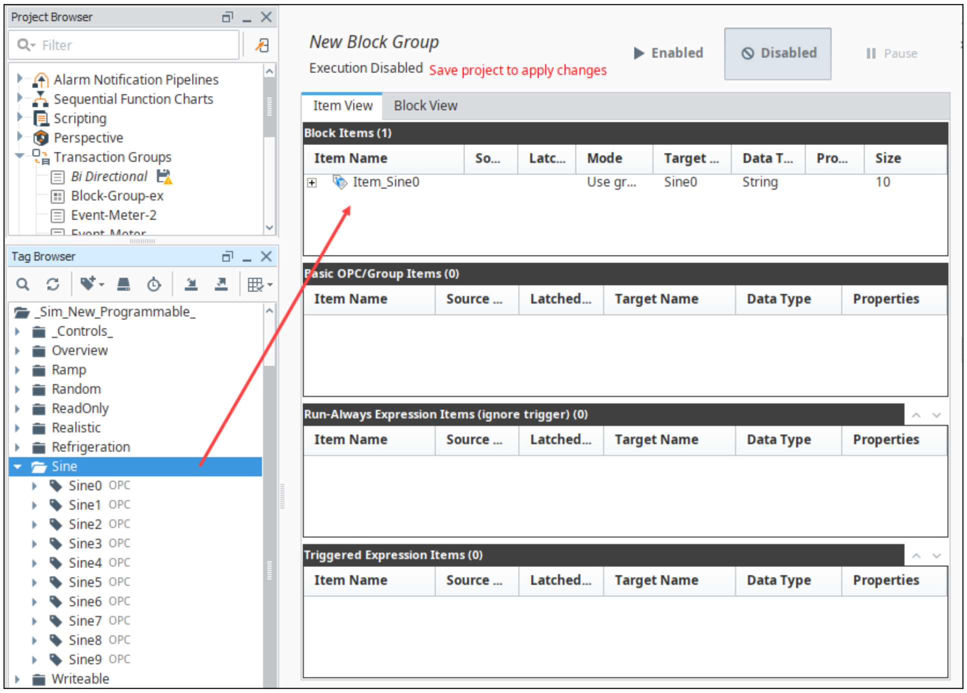
Watch the Video

Create a Block Group

1. In the Project Browser, right-click on Transaction Groups and select **New Transaction Group > New Block Group**.



2. Give the group a name and click **Create Group**.
3. Drag a Tag folder into the **Block Items** section of the new Transaction Group.



Block Items (1)

Item Name	So...	Latc...	Mode	Target ...	Data T...	Pro...	Size
Item_Sine0			Use gr...	Sine0	String		10

Basic OPC/Group Items (0)

Item Name	Source ...	Latched...	Target Name	Data Type	Properties

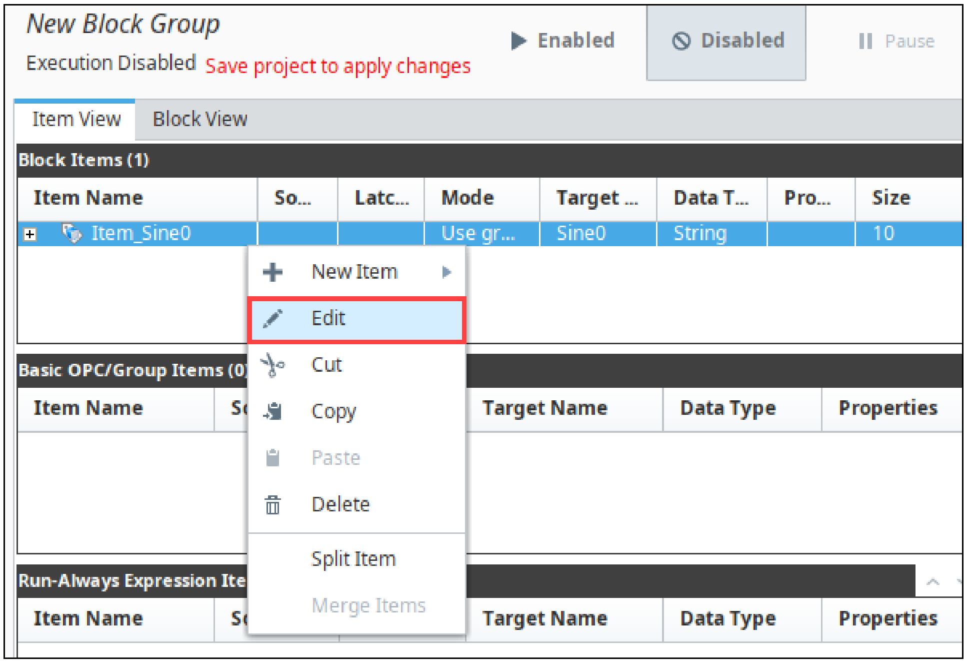
Run-Always Expression Items (ignore trigger) (0)

Item Name	Source ...	Latched...	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source ...	Latched...	Target Name	Data Type	Properties

4. Select the item in the Block group, right-click and select **Edit**.



Block Items (1)

Item Name	So...	Latc...	Mode	Target ...	Data T...	Pro...	Size
Item_Sine0			Use gr...	Sine0	String		10

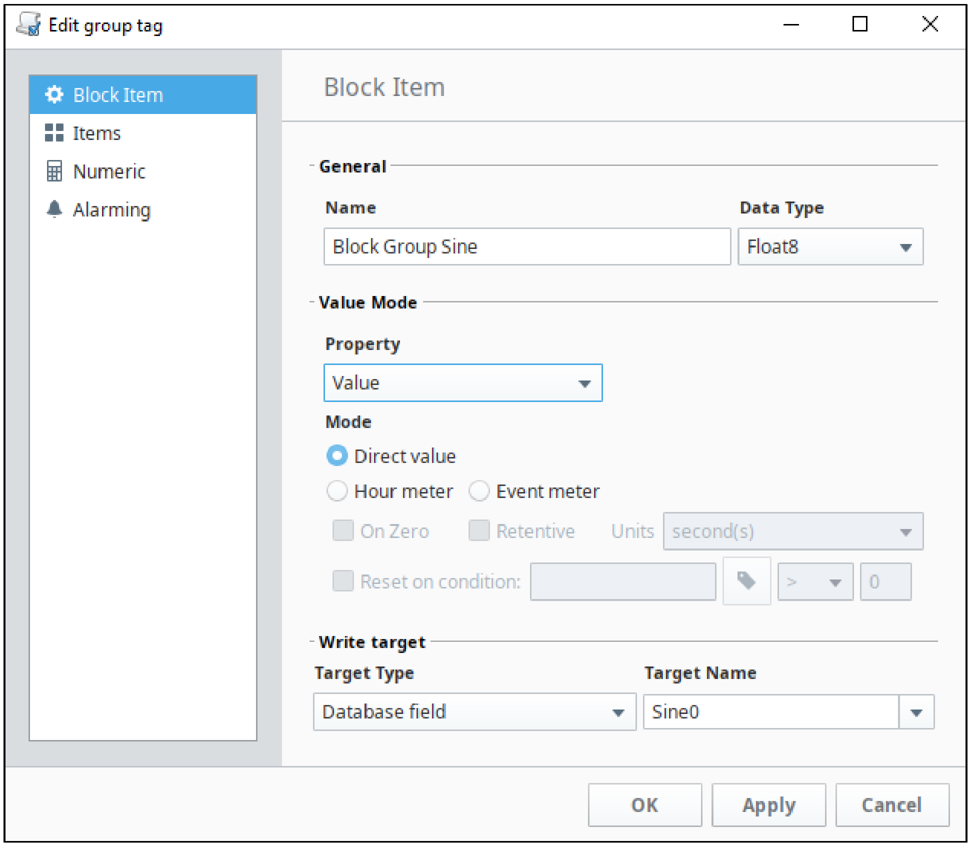
Basic OPC/Group Items (0)

Item Name	S	Target Name	Data Type	Properties

Run-Always Expression Ite

Item Name	S	Target Name	Data Type	Properties

5. Change the **Name**, enter the **Target Name** to anything appropriate. Click **OK**.



Block Item

General

Name: Block Group Sine

Data Type: Float8

Value Mode

Property: Value

Mode

● Direct value

○ Hour meter ○ Event meter

□ On Zero □ Retentive Units second(s)

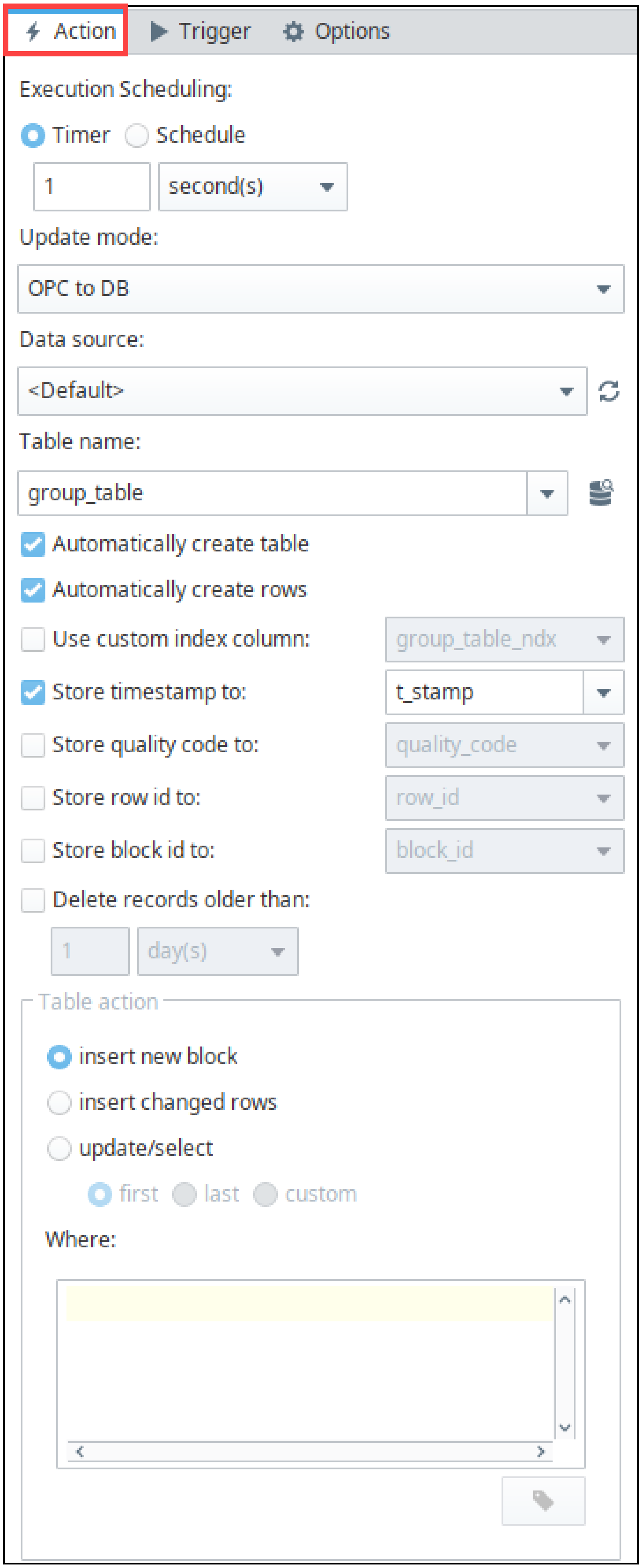
□ Reset on condition: > 0

Write target

Target Type: Database field

Target Name: Sine0

6. Configure the remainder of the group settings under the **Action** tab.



Execution Scheduling:

● Timer ○ Schedule

1 second(s)

Update mode:

OPC to DB

Data source:

<Default>

Table name:

group_table

☑ Automatically create table

☑ Automatically create rows

□ Use custom index column: group_table_ndx

☑ Store timestamp to: t_stamp

□ Store quality code to: quality_code

□ Store row id to: row_id

□ Store block id to: block_id

□ Delete records older than:

1 day(s)

Table action

● insert new block

○ insert changed rows

○ update/select

● first ○ last ○ custom

Where:

7. Select the group, and click **Enabled**.
8. **Save** the project to start the group.

DB to OPC Mode with Custom Where Clause

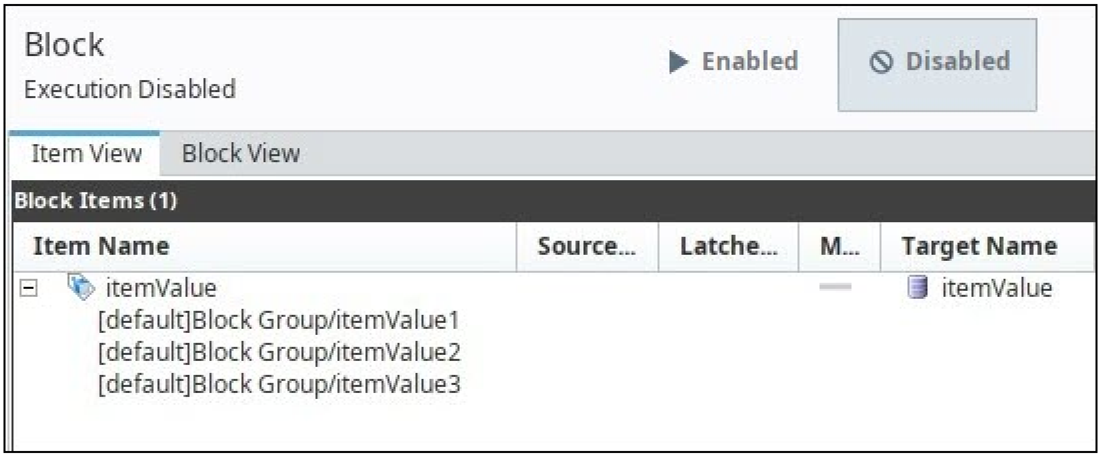
Like the Standard Group, block groups can be configured to retrieve records from the database, writing back to an OPC address or Tag. When using a custom WHERE clause, you can write the WHERE statement in such a way that multiple rows are returned, which would then update multiple items, which in turn write back to to OPC addresses. We could then add a dynamic OPC value as a "lookup" that would determine which set of rows to return.

This is a great way to retrieve multiple datapoints that are stored in a tall format on a database table, ideally when you're looking to retrieve multiple sequential rows. For example a table with the following content, a single block item targeting the "itemValue" column, and a "lookup" Tag or OPC item that the group will use in the WHERE clause.

Table structure

table_ndx	itemValue
1	1
2	20
3	300
4	4,000
5	50,000
6	600,000
7	7,000,000

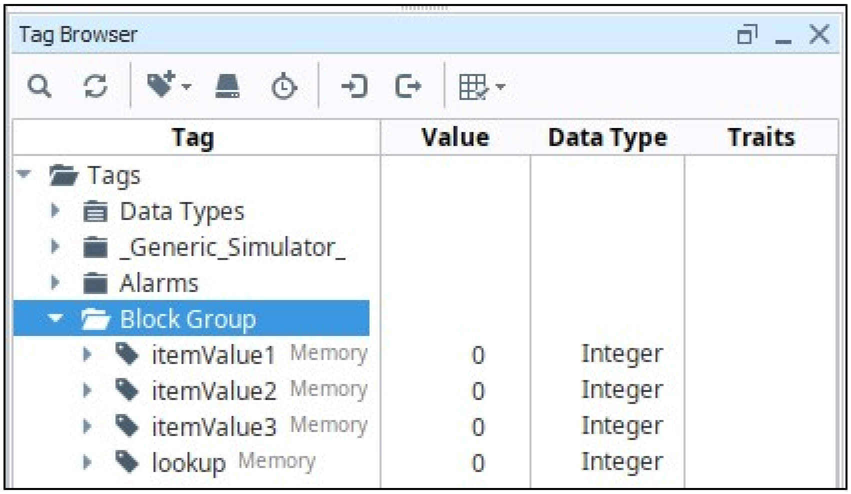
Our block item



Block Items (1)

Item Name	Source...	Latche...	M...	Target Name
itemValue [default]Block Group/itemValue1 [default]Block Group/itemValue2 [default]Block Group/itemValue3			—	itemValue

Our Tags, including "lookup"



Tag Browser

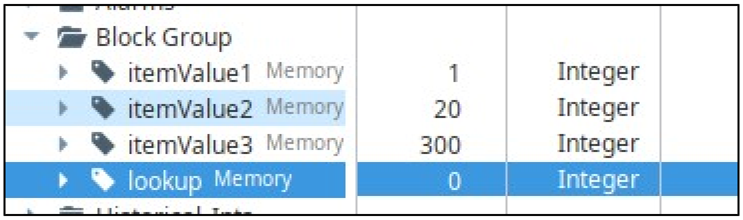
Tag	Value	Data Type	Traits
Tags			
Data Types			
_Generic_Simulator_			
Alarms			
Block Group			
itemValue1 Memory	0	Integer	
itemValue2 Memory	0	Integer	
itemValue3 Memory	0	Integer	
lookup Memory	0	Integer	

1. Set the **"Update mode"** for the group to **"DB to OPC."**
2. Set the Table action (under the "Action" tab) to **"update/select."**
3. Select the **"custom"** radio button.
4. Under the **"Where:"** text area, click the Tag icon, and select the **"lookup"** Tag, which adds a reference to the Tag like this: {[default]Block Group/lookup}
5. Write the rest of the our condition. In this case, we'll say we want results from our table starting a value greater than our lookup value. Using the table specified above, we could write the following condition:

```
null_table_test_ndx > {[default]Block Group/lookup}
```

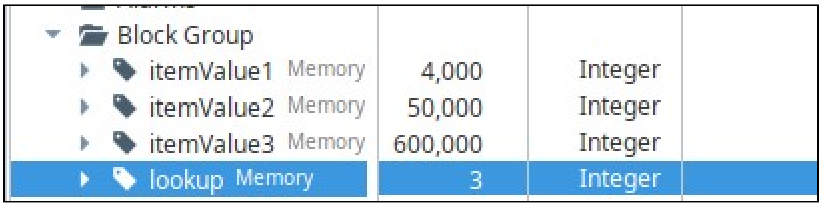
6. **Enable** the group, and **save** the project.

When the group is running, with an initial lookup value of 0, the group automatically grab table_ndx of 1, and write a value of 1 (from the first row) to itemValue1, a value of 20 (from the second row) to itemValue2, and so on.



Block Group		
itemValue1 Memory	1	Integer
itemValue2 Memory	20	Integer
itemValue3 Memory	300	Integer
lookup Memory	0	Integer

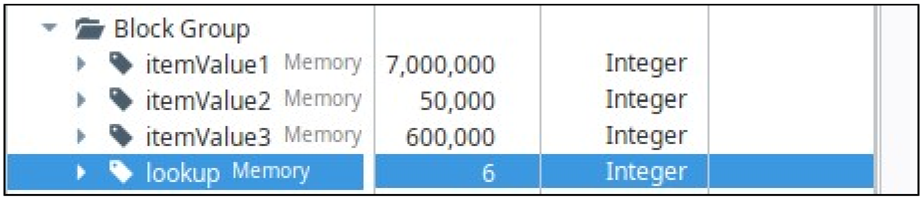
If we set the value on lookup to 3, that means the first row in the result set will be row 4, setting itemValue1 to 4000, itemValue2 to 50,000, and so on.



Block Group		
itemValue1 Memory	4,000	Integer
itemValue2 Memory	50,000	Integer
itemValue3 Memory	600,000	Integer
lookup Memory	3	Integer

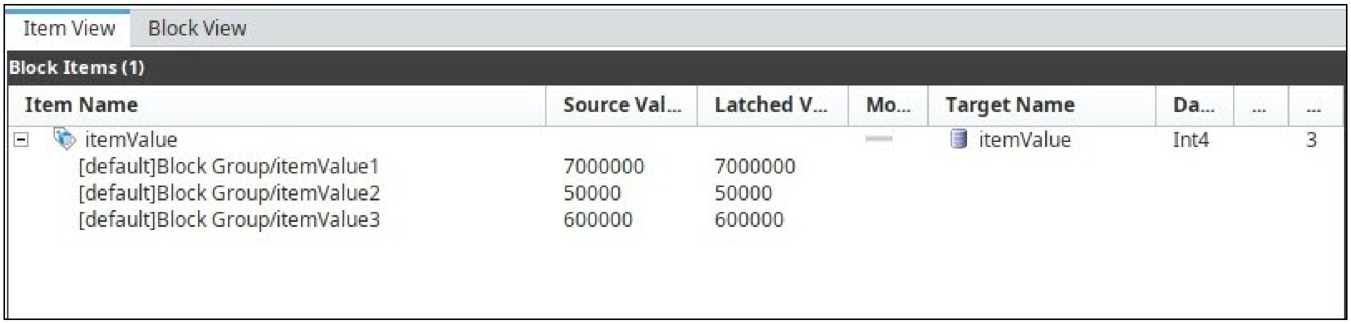
If you set the value of lookup to 6, then that will set the value on itemValue1 to row 7's value (7,000,000), but you'll notice the other Tags are retaining a value, which is notable since those items don't have a corresponding value to retrieve.

Values on our Tags



Block Group		
itemValue1 Memory	7,000,000	Integer
itemValue2 Memory	50,000	Integer
itemValue3 Memory	600,000	Integer
lookup Memory	6	Integer

Items in the group



Block Items (1)

Item Name	Source Val...	Latched V...	Mo...	Target Name	Da...	...	...
⊟ itemValue			—	itemValue	Int4		3
[default]Block Group/itemValue1	7000000	7000000					
[default]Block Group/itemValue2	50000	50000					
[default]Block Group/itemValue3	600000	600000					

This is expected. By default, when a Block Group is configured like this, and some items can't receive updated values as a result of the dynamic WHERE clause not returning enough rows, the items will retain their previous latched value: that is to say, the group will not automatically clear or reset the values on the other items. Refer to Set NULL DB Values to Default.

Next...

- Recipe Group

Recipe Group

You can use Transaction Groups to create a recipe management system which will pull recipe information from the database and push it to the PLC when requested. With this system, the Transaction Group is what queries the database rather than writing scripts to handle it all.

Recipe Group

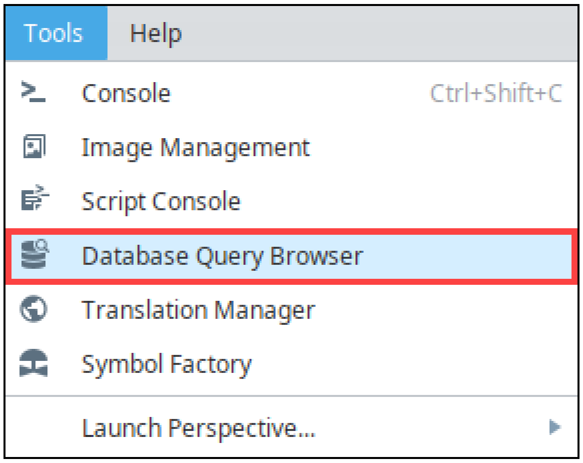
Watch the Video

Before the Transaction Group

Before we make the Transaction Group, we first need to make sure we have a table set up in our database that holds recipes. If you already have this, then you can skip to the next step on making the Transaction Group.

We will make a table in our database that will hold our recipes. Our recipes will be simple, containing a name, unique id, and two setpoints, so we will need a column for each of those values.

1. Verify the Designer's **Comm Mode** is set to Read/Write, and open up the **Database Query Browser**.



2. **Execute** the query below in the Database Query Browser to create the table we'll use in this example:

```
CREATE TABLE recipes(
        id INT PRIMARY KEY,
        recipe_name VARCHAR(50),
        setpoint1 FLOAT,
        setpoint2 FLOAT)
```

Note: This query was designed for an MSSQL database. If you are connected to a different database, the syntax on the CREATE statement may differ. Check your database's documentation for more details.

3. Next we need to put some data into the table by using an `insert` statement. Execute the below query to insert a new record into our recipes table:

```
INSERT INTO recipes (id, recipe_name, setpoint1, setpoint2)
        VALUES (1, 'Recipe 1', 10, 0)
```

You can rerun this query as many times as you want, incrementing the id to give you a new unique id, changing the name, and providing different setpoints. Your table might look something like the one below.

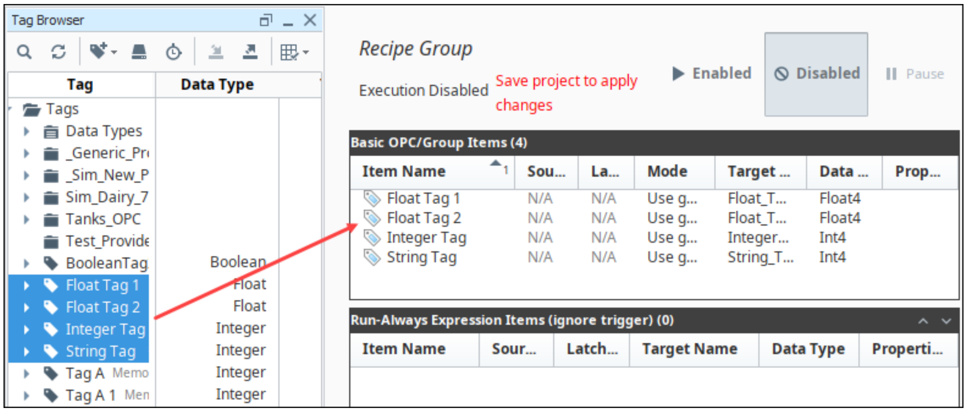
id	recipe_name	setpoint1	setpoint2
1	The First Recipe	34.7	54.1
2	The Wrong Recipe	12.8	42.3

3	The Best Recipe	65.7	95.1
4	The Other Recipe	49.8	112.2

Create the Transaction Group

Now that we have a recipe table in the database that is populated with some records, we can create the Transaction Group that will load a recipe from the table into our Tags. We will be using the recipes table that we put together previously, but if you already had a table, you can use that here instead.

1. Create a new **Standard Transaction Group**.
2. We have four columns in our database table, so we will need four Tags to use in the Transaction Group: an integer, string, and two floats for the id, name, and setpoints respectively. Add the four Tags to the Transaction Group.



Tag	Data Type
Tags	
Data Types	
_Generic_Pr	
_Sim_New_P	
Sim_Dairy_7	
Tanks_OPC	
Test_Provide	
BooleanTag	Boolean
Float Tag 1	Float
Float Tag 2	Float
Integer Tag	Integer
String Tag	Integer
Tag A Memo	Integer
Tag A 1 Men	Integer

Recipe Group

Execution Disabled

Save project to apply changes

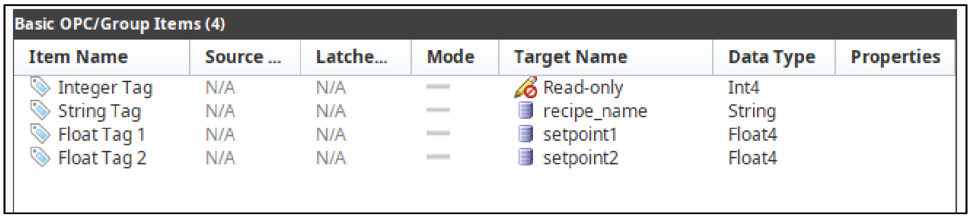
Basic OPC/Group Items (4)

Item Name	Sou...	La...	Mode	Target ...	Data ...	Prop...
Float Tag 1	N/A	N/A	Use g...	Float_T...	Float4	
Float Tag 2	N/A	N/A	Use g...	Float_T...	Float4	
Integer Tag	N/A	N/A	Use g...	Integer...	Int4	
String Tag	N/A	N/A	Use g...	String_T...	Int4	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Sour...	Latch...	Target Name	Data Type	Properti...

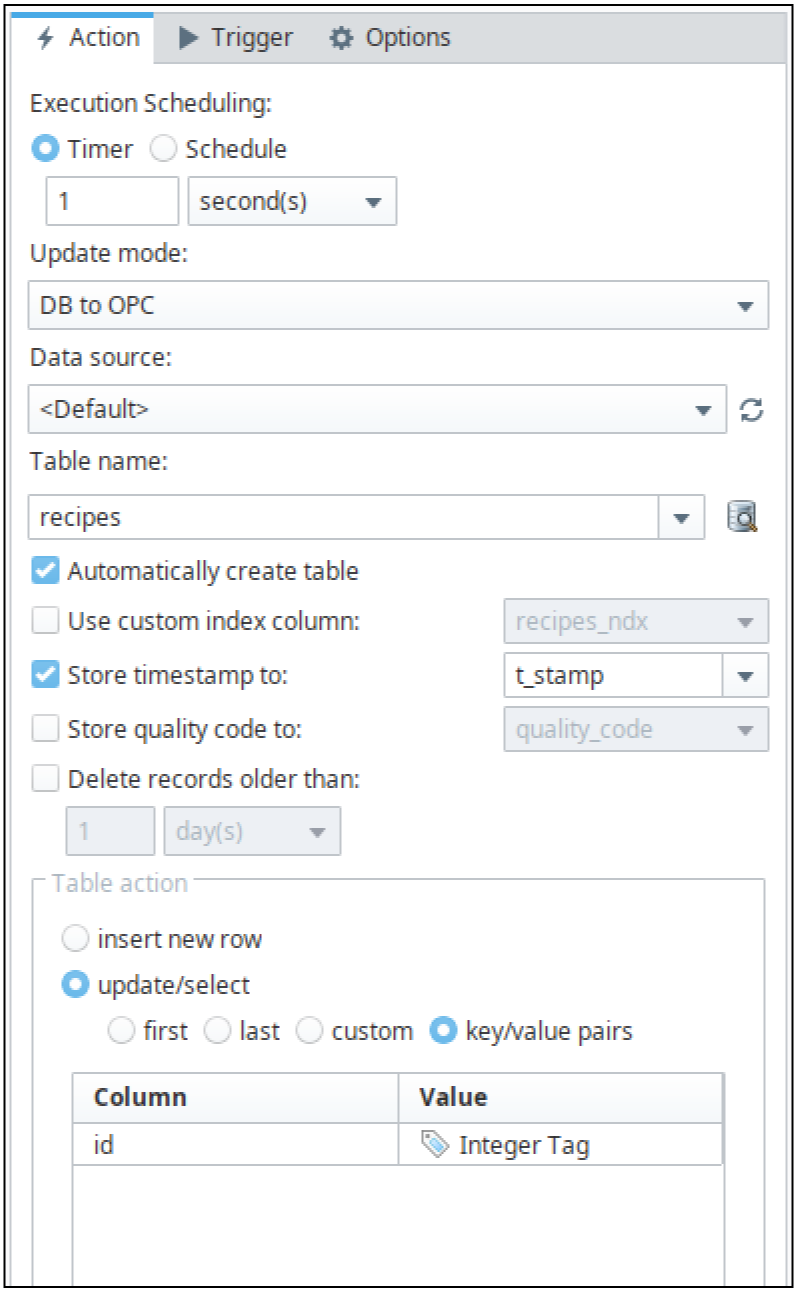
3. Set the Table Name to '**recipes**', the table that we created earlier.
4. We then need to ensure that our Tags will be receiving the proper values from the database.
   a. Set the Target Names for each of the Tags: the string to '**recipe_name**', the floats to **'setpoint1'** and **'setpoint2'**
   b. Set integer to **'Read Only'**. We don't need to set the integer to the id column, because we will not pull the the id from the database, but rather use the id as a trigger and in the where clause.



Basic OPC/Group Items (4)

Item Name	Source ...	Latche...	Mode	Target Name	Data Type	Properties
Integer Tag	N/A	N/A	—	Read-only	Int4	
String Tag	N/A	N/A	—	recipe_name	String	
Float Tag 1	N/A	N/A	—	setpoint1	Float4	
Float Tag 2	N/A	N/A	—	setpoint2	Float4	

5. Now we can finish setting up the rest of the Transaction Group. Set the Update mode to **DB** to **OPC**.
6. Set the **Table Action** to **Update/Select** using **Key/Value Pairs** with the **Column** set to **id**, and the **Value** set to the **Integer Tag** you are using.



Execution Scheduling:

● Timer ○ Schedule

1 second(s)

Update mode: DB to OPC

Data source: <Default>

Table name: recipes

☑ Automatically create table

□ Use custom index column: recipes_ndx

☑ Store timestamp to: t_stamp

□ Store quality code to: quality_code

□ Delete records older than: 1 day(s)

Table action

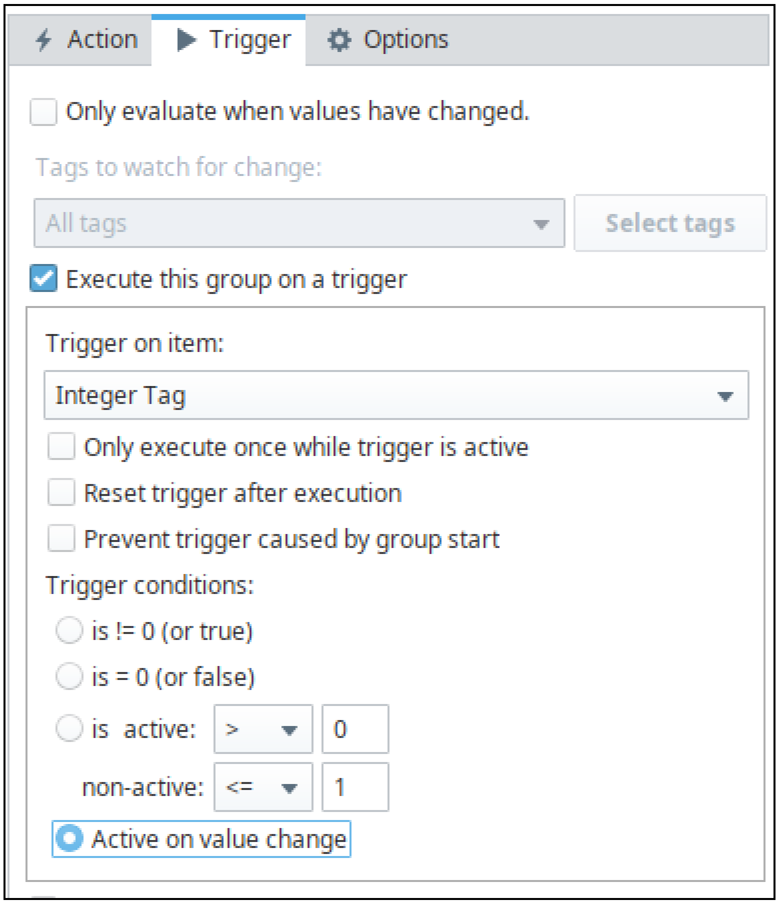
○ insert new row

● update/select

○ first ○ last ○ custom ● key/value pairs

Column	Value
id	Integer Tag

7. Set the Update Rate to **1 second**. We want to query the values out of the database as soon as we ask for them, so we need the group to update quickly. However, we don't want the group to actually query the database every second, so we will need to set up the trigger.
8. Go to the Trigger tab, and select **Execute this group on a trigger**. Trigger on the item the int Tag that is being used for the id. Specify the Trigger condition as **Active on value change**.



□ Only evaluate when values have changed.

Tags to watch for change: All tags

☑ Execute this group on a trigger

Trigger on item: Integer Tag

□ Only execute once while trigger is active

□ Reset trigger after execution

□ Prevent trigger caused by group start

Trigger conditions:

○ is != 0 (or true)

○ is = 0 (or false)

○ is active: > 0

non-active: <= 1

● Active on value change

9. Finally, **Enable** the Transaction Group and **save** the project to get it started. The Transaction Group will now pull the recipe out of the database where the id matches the value of the int Tag. The trigger also prevents it from running all the time, instead running only when the int Tag value changes.
10. To test it out, simply change the value of id Tag to an id of one of the recipes in the recipes table.

Update or Insert Group

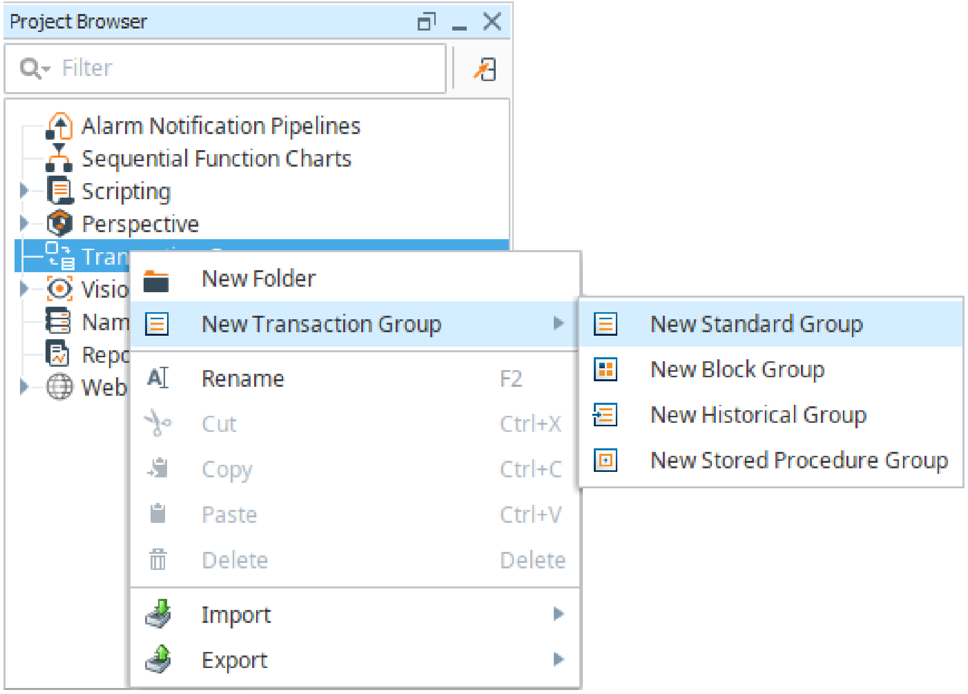
You can update a row or insert a new row into the database when a key pair combination does not exist. This eliminates the need to have a database that has every possible option considered in its original design. Because of the **insert row when not present** setting, the group will insert a new record whenever the designated ID doesn't exist. Afterwards, it will update the rows in the table that are associated with the key/value references as shown in this example.

Update or Insert Group

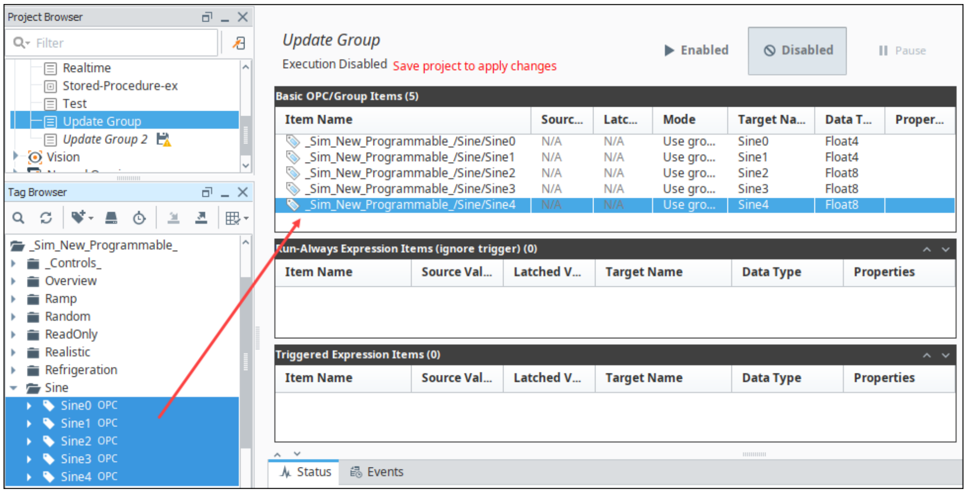
Watch the Video

Update or Insert a New Row into the Database

1. In the Project Browser, right-click on Transaction Groups and select **New Standard Group**.



2. Give the group a name and click **Create Group**.
3. Drag a group of Tags into the groups **Basic OPC/Group Items** section.



Update Group

Execution Disabled Save project to apply changes

Basic OPC/Group Items (5)

Item Name	Sourc...	Latc...	Mode	Target Na...	Data T...	Proper...
_Sim_New_Programmable_/Sine/Sine0	N/A	N/A	Use gro...	Sine0	Float4	
_Sim_New_Programmable_/Sine/Sine1	N/A	N/A	Use gro...	Sine1	Float4	
_Sim_New_Programmable_/Sine/Sine2	N/A	N/A	Use gro...	Sine2	Float8	
_Sim_New_Programmable_/Sine/Sine3	N/A	N/A	Use gro...	Sine3	Float8	
_Sim_New_Programmable_/Sine/Sine4	N/A	N/A	Use gro...	Sine4	Float8	

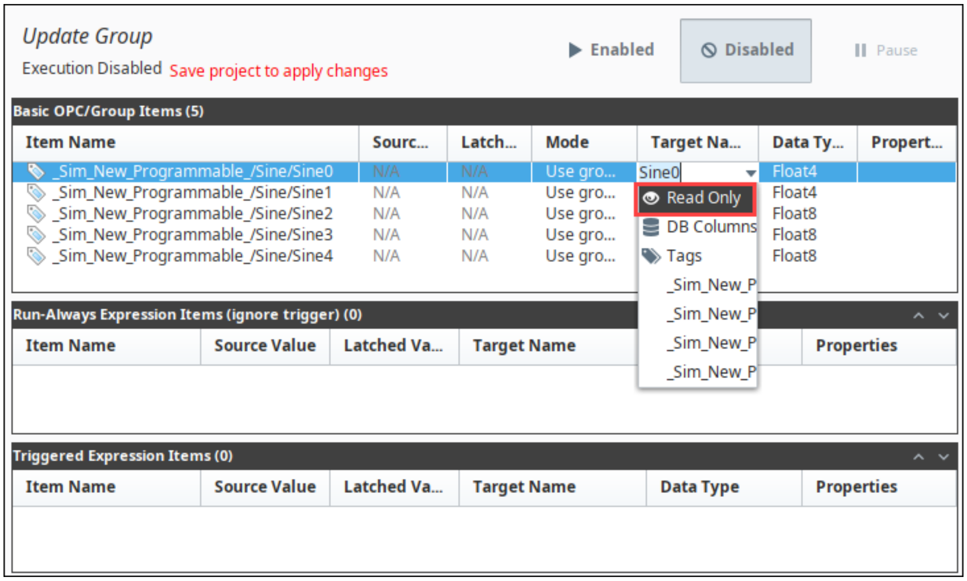
Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Val...	Latched V...	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source Val...	Latched V...	Target Name	Data Type	Properties

4. Change one of the Tags to be read-only by selecting **Read Only** from the Tag's **Target Name** column.



Update Group

Execution Disabled Save project to apply changes

Basic OPC/Group Items (5)

Item Name	Sourc...	Latch...	Mode	Target Na...	Data Ty...	Propert...
_Sim_New_Programmable_/Sine/Sine0	N/A	N/A	Use gro...	Sine0	Float4	
_Sim_New_Programmable_/Sine/Sine1	N/A	N/A	Use gro...		Float4	
_Sim_New_Programmable_/Sine/Sine2	N/A	N/A	Use gro...		Float8	
_Sim_New_Programmable_/Sine/Sine3	N/A	N/A	Use gro...		Float8	
_Sim_New_Programmable_/Sine/Sine4	N/A	N/A	Use gro...		Float8	

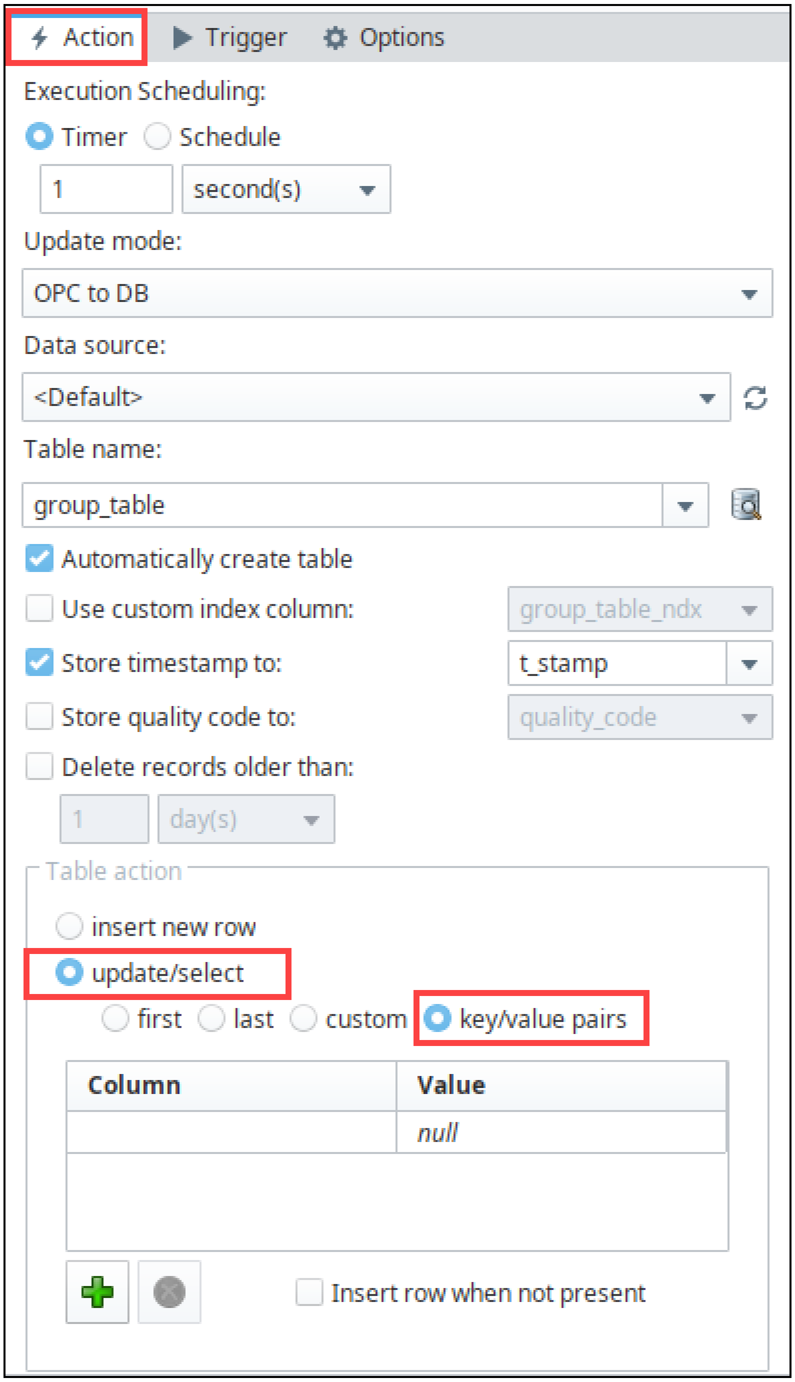
Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Value	Latched Va...	Target Name		Properties

Triggered Expression Items (0)

Item Name	Source Value	Latched Va...	Target Name	Data Type	Properties

5. In the group's **Action** tab, in the **Table action** area, select the **update/select** radio button and the **key/value pairs** radio button.



Execution Scheduling:

● Timer ○ Schedule

1 second(s)

Update mode:

OPC to DB

Data source:

<Default>

Table name:

group_table

☑ Automatically create table

□ Use custom index column: group_table_ndx

☑ Store timestamp to: t_stamp

□ Store quality code to: quality_code

□ Delete records older than:

1 day(s)

Table action

○ insert new row

● update/select

○ first ○ last ○ custom ● key/value pairs

Column	Value
	null

□ Insert row when not present

6. Click the **Add** ✚ icon.
   a. For the **Column** select the database table ID column.
   b. For the **Value** column, select the read-only Tag.
   c. Select **Insert row when not present** check box at the bottom of the **Table action** area.

7. Select the group, and click **Enabled**.
8. Save the project to start the group.

Next...

Trigger Options

It is often useful to execute a group only when a certain condition is met or as a bit turns on or off. Triggers allow Transaction Groups to run based on values changing in various ways.

On this page ...

INDUCTIVE UNIVERSITY

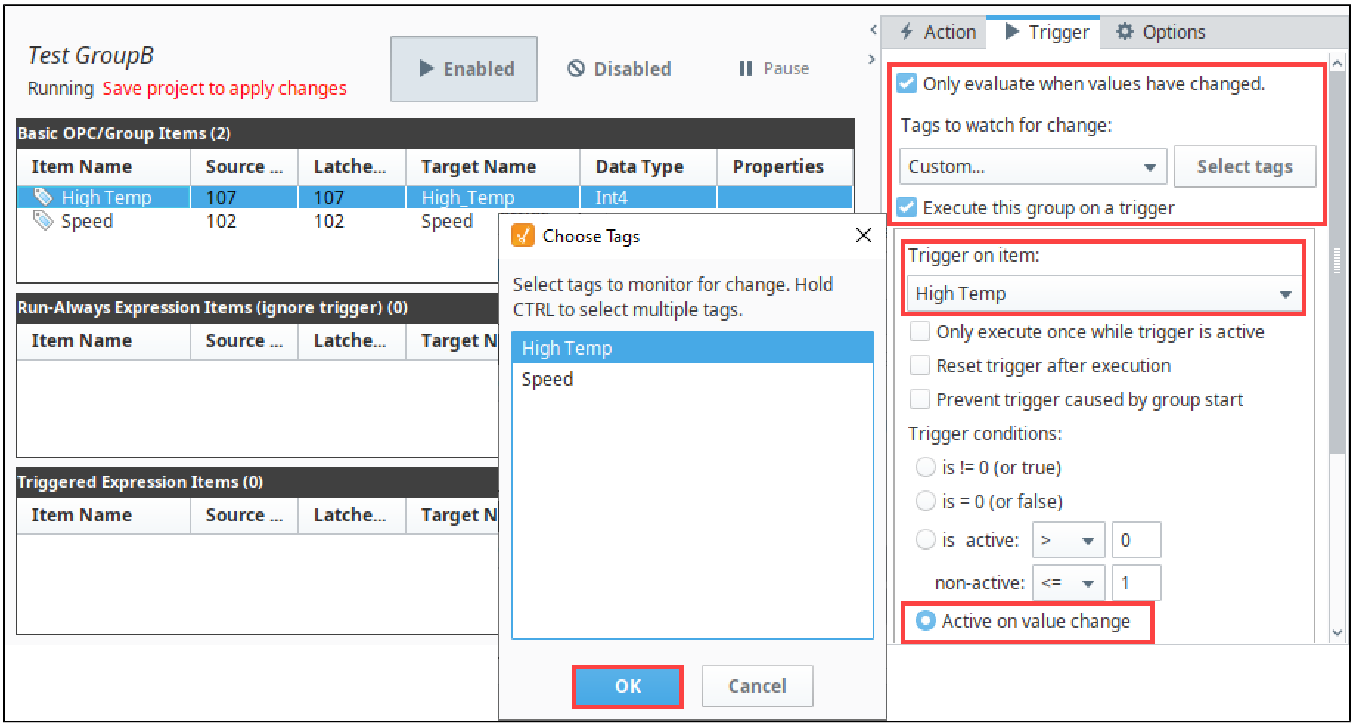
Trigger – On Value Change

Watch the Video

Execute on Value Change

A group can execute when the group's Tags have changed, or when a particular Tag within the group has changed. In either case, the Transaction Group will execute every time the value or values change.

1. In the **Trigger** tab, select at the very top the **Only evaluate when values have changed** checkbox.
   Now the group will execute if any of the Tags change.
2. To execute when only one Tag changes, from the **Tags to watch for change** dropdown, select **Custom**, click on **Select Tags**, select the Tag from the pop-up window, and click **OK**. You can select more than one Tag at a time in order to monitor more than one Tag for value changes.
3. From the **Trigger on item** dropdown, select the appropriate Tag to execute the Tag on a trigger.
4. Select the **Active on value change** radio button.
5. **Save** the Project to start the Transaction Group.



Basic OPC/Group Items (2)

Item Name	Source ...	Latche...	Target Name	Data Type	Properties
High Temp	107	107	High_Temp	Int4	
Speed	102	102	Speed		

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source ...	Latche...	Target N

Triggered Expression Items (0)

Item Name	Source ...	Latche...	Target N

Select tags to monitor for change. Hold CTRL to select multiple tags.

- High Temp
- Speed

☑ Only evaluate when values have changed.

Tags to watch for change: Custom... Select tags

☑ Execute this group on a trigger

Trigger on item: High Temp

□ Only execute once while trigger is active

□ Reset trigger after execution

□ Prevent trigger caused by group start

Trigger conditions:

○ is != 0 (or true)

○ is = 0 (or false)

○ is active: > 0

non-active: <= 1

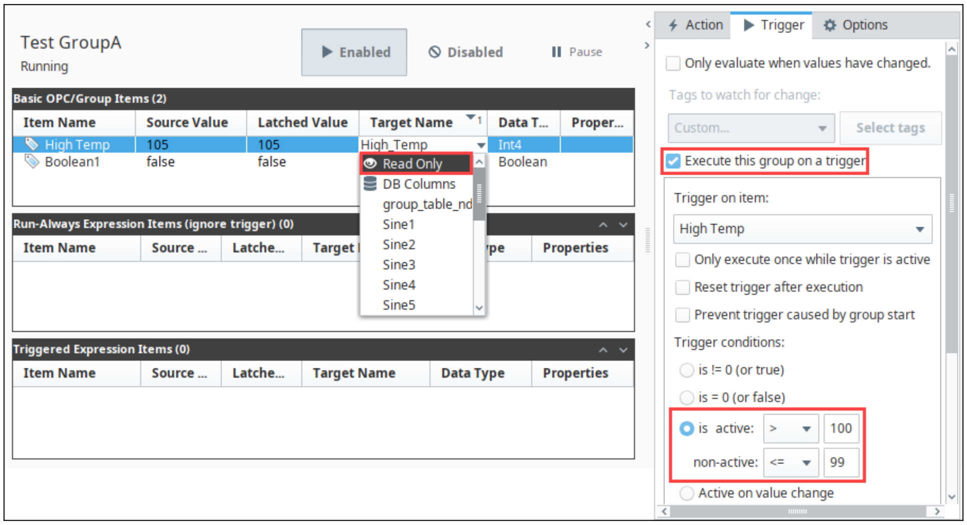
● Active on value change

Execute while Condition Is True

Groups can execute while a condition is true resulting in the Transaction Group continuing to execute for the duration of this condition.

1. Create a Transaction Group, and drag a numeric or boolean Tag into the **Basic OPC/Group Items** section.

2. From the **Target Name** column dropdown, select **Read Only** if you do not want the trigger value to be written to the database.
3. Go to the **Trigger** tab, and select the **Execute this group on a trigger** checkbox.
4. In the **Trigger conditions** area, set the trigger conditions which will determine under what condition the group executes.
5. **Save** the Project to start the Transaction Group.



Basic OPC/Group Items (2)

Item Name	Source Value	Latched Value	Target Name	Data T...	Proper...
High Temp	105	105	High_Temp	Int4	
Boolean1	false	false		Boolean	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source ...	Latche...	Target I	...pe	Properties

Triggered Expression Items (0)

Item Name	Source ...	Latche...	Target Name	Data Type	Properties

□ Only evaluate when values have changed.

Tags to watch for change: Custom... Select tags

☑ Execute this group on a trigger

Trigger on item: High Temp

□ Only execute once while trigger is active

□ Reset trigger after execution

□ Prevent trigger caused by group start

Trigger conditions:

○ is != 0 (or true)

○ is = 0 (or false)

● is active: > 100

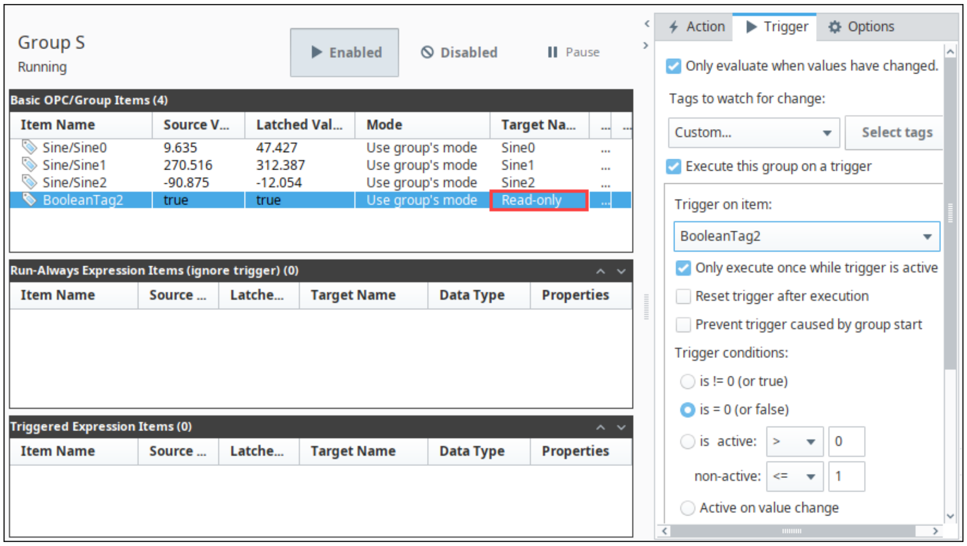
non-active: <= 99

○ Active on value change

Execute on a Rising Edge

Groups can execute when the trigger becomes True. This is known as a **rising edge trigger** and it will only execute once and will not execute again until the trigger repeats the same cycle.

1. Create a standard Transaction Group with any number of Tags as long as one of them is a boolean Tag that will serve at the trigger for the group.
2. Set the **Write Target** for the boolean Tag to **Read-only** by selecting read-only from its drop down in the Target Name column. You do not need to do this if you want the trigger value to be written to the database.
3. Go to the **Trigger** tab and select the check box to **Execute this group on a trigger**. Select the boolean Tag from the drop down menu and select to have the group only **execute once while the trigger is active**.
4. **Save** the Project to start the Transaction Group.



Basic OPC/Group Items (4)

Item Name	Source V...	Latched Val...	Mode	Target Na...	...	...
Sine/Sine0	9.635	47.427	Use group's mode	Sine0	...	
Sine/Sine1	270.516	312.387	Use group's mode	Sine1	...	
Sine/Sine2	-90.875	-12.054	Use group's mode	Sine2	...	
BooleanTag2	true	true	Use group's mode	Read-only	...	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source ...	Latche...	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source ...	Latche...	Target Name	Data Type	Properties

☑ Only evaluate when values have changed.

Tags to watch for change: Custom... Select tags

☑ Execute this group on a trigger

Trigger on item: BooleanTag2

☑ Only execute once while trigger is active

□ Reset trigger after execution

□ Prevent trigger caused by group start

Trigger conditions:

○ is != 0 (or true)

● is = 0 (or false)

○ is active: > 0

non-active: <= 1

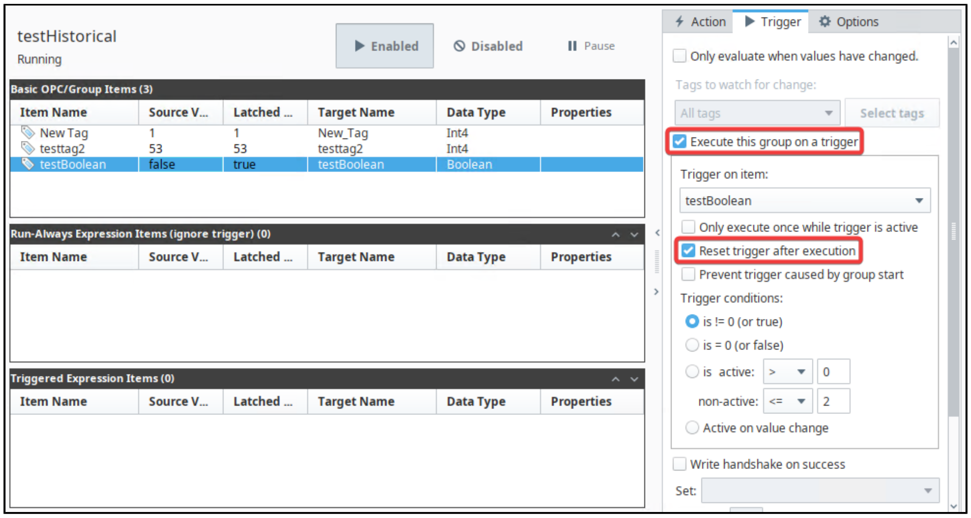
○ Active on value change

Reset Trigger

Resetting a trigger after execution of a triggered Transaction Group will result in the Transaction Group writing once to the targets followed by writing back to the trigger to reset it.

To reset the trigger after execution:

1. Create a Transaction Group with a boolean Tag. The Target Name column for the trigger Tag can be read-only, though this is not mandatory.
2. Select the **Trigger** tab and select the **Execute this group on a trigger** check box.
3. Select the **Reset trigger after execution** check box.
4. **Save** the Project to start the Transaction Group.



testHistorical

Running

▶ Enabled ⦸ Disabled ⏸ Pause

Basic OPC/Group Items (3)

Item Name	Source V...	Latched ...	Target Name	Data Type	Properties
New Tag	1	1	New_Tag	Int4	
testtag2	53	53	testtag2	Int4	
testBoolean	false	true	testBoolean	Boolean	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source V...	Latched ...	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source V...	Latched ...	Target Name	Data Type	Properties

Action | Trigger | Options

□ Only evaluate when values have changed.

Tags to watch for change: All tags | Select tags

☑ Execute this group on a trigger

Trigger on item: testBoolean

□ Only execute once while trigger is active

☑ Reset trigger after execution

□ Prevent trigger caused by group start

Trigger conditions:

● is != 0 (or true)

○ is = 0 (or false)

○ is active: > 0

non-active: <= 2

○ Active on value change

□ Write handshake on success

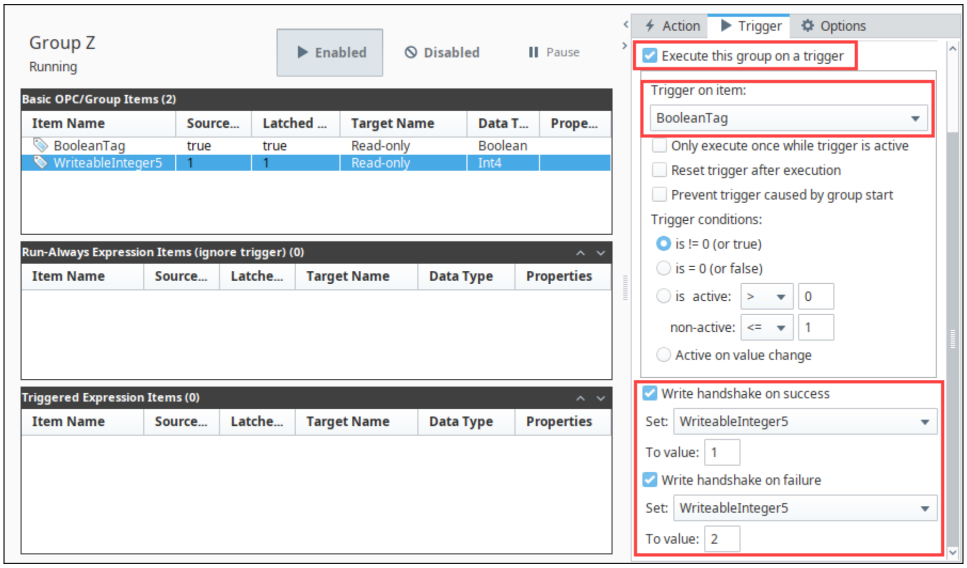
Set:

Handshakes

When a group executes, it either completes successfully or an error prevents its execution. The outcome of an execution can be handled in the handshake section of the trigger section of the Transaction Group. When a group executes successfully or fails to execute, the handshake can write a value back to a Tag to alert the user that the group executed successfully or unsuccessfully.

To set handshake values for alerting the user:

1. Create a Transaction Group with a boolean Tag and a numeric Tag.
2. Set the boolean and the numeric Tag to read only.
3. Go to the **Trigger** tab and choose to **Execute this group on a trigger**.
4. Select the boolean Tag as the trigger In the **Trigger on item** drop down, and select the appropriate execution conditions.
5. In the bottom section, select **Write handshake on success**, select the numeric Tag to write to, and choose a number that signifies success.
6. Likewise, in the bottom section, select **Write handshake on failure**, select the numeric Tag to write to, and choose a number that signifies failure.
7. **Save** the Project to start the Transaction Group.



Group Z

Running

▶ Enabled ⦸ Disabled ⏸ Pause

Basic OPC/Group Items (2)

Item Name	Source...	Latched ...	Target Name	Data T...	Prope...
BooleanTag	true	true	Read-only	Boolean	
WriteableInteger5	1	1	Read-only	Int4	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source...	Latche...	Target Name	Data Type	Properties

Triggered Expression Items (0)

Item Name	Source...	Latche...	Target Name	Data Type	Properties

Action | Trigger | Options

☑ Execute this group on a trigger

Trigger on item: BooleanTag

□ Only execute once while trigger is active

□ Reset trigger after execution

□ Prevent trigger caused by group start

Trigger conditions:

● is != 0 (or true)

○ is = 0 (or false)

○ is active: > 0

non-active: <= 1

○ Active on value change

☑ Write handshake on success

Set: WriteableInteger5

To value: 1

☑ Write handshake on failure

Set: WriteableInteger5

To value: 2

Next...

- Understanding Transaction Groups
- Transaction Group Examples

Transaction Group Update Modes

On this page ...

Transaction Groups are generally used to store OPC data into a database. Transaction Group Update Modes give users additional flexibility as to whether data should flow from an OPC server to a database or from a database to an OPC server. Additionally, it is possible to configure data to be synchronized between a database and an OPC server via Bi-directional Update Modes.

All update modes do not work for all Transaction Group types. For example, Historical Transaction Groups can only insert data to a database table and not update it. In addition, Historical Transaction Groups also cannot write back to OPC items so Bi-directional Update Mode will not be an option for users using the Historical Transaction Group type.

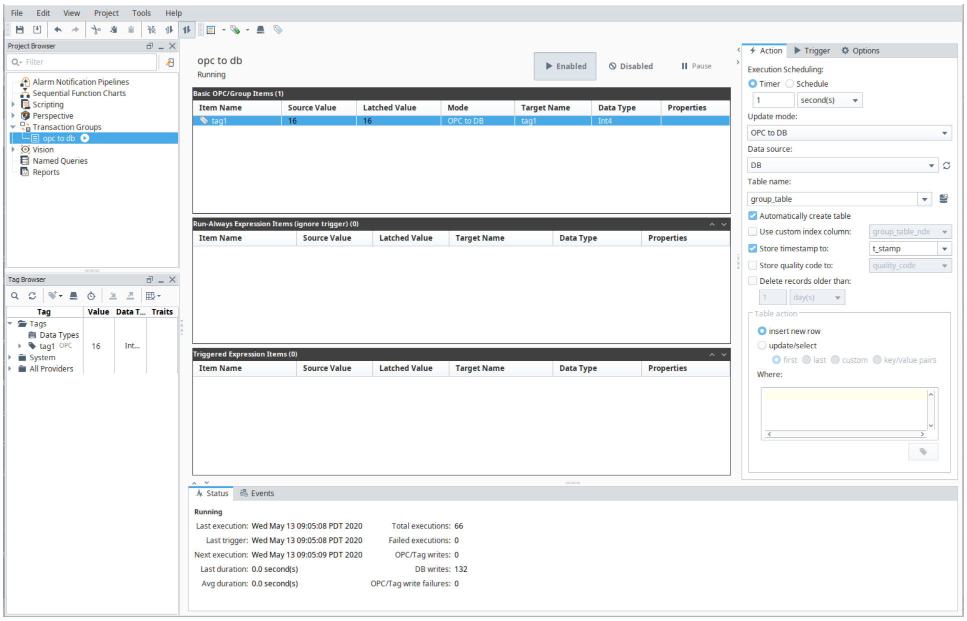
The different Update Modes:

- OPC to DB - Only read from the OPC server and write to the database.
- DB to OPC - Only read from the database and write to the OPC Server.
- Bi-directional OPC wins - Read and Write to both the database and OPC Server. On group start, write OPC values to the database.
- Bi-directional DB wins - Read and Write to both the database and OPC Server. On group start, write database values to OPC items.

OPC to DB

The **OPC to DB** Update Mode allows a Transaction Group to store OPC data to a Database Ignition that it has a connection to as shown in the following example.

1. Create a Standard Transaction Group and from your Tag Browser, drag a single tag into your Transaction Group's Basic OPC/Group Items section. For this example, the tag is called 'tag1.'
2. Set the Update Mode on 'tag1' to **OPC to DB** and set its **Target Name** to be 'tag1.' The Target Name will correlate to the name of the column in your database table where 'tag1' will be stored.



Tag	Value	Data T...	Traits
Tags			
Data Types			
tag1 OPC	16	Int...	
System			
All Providers			

opc to db
Running

Basic OPC/Group Items (1)

Item Name	Source Value	Latched Value	Mode	Target Name	Data Type	Properties
tag1	16	16	OPC to DB	tag1	Int4	

Run-Always Expression Items (ignore trigger) (0)

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties

Triggered Expression Items (0)

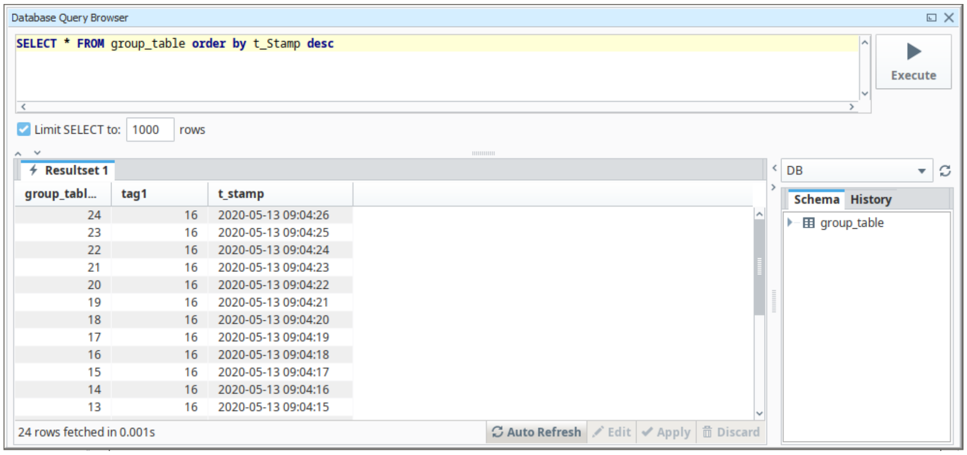
Item Name	Source Value	Latched Value	Target Name	Data Type	Properties

Execution Scheduling:
● Timer ○ Schedule
1 second(s)
Update mode: OPC to DB
Data source: DB
Table name: group_table
☑ Automatically create table
□ Use custom index column: group_table_ndx
☑ Store timestamp to: t_stamp
□ Store quality code to: quality_code
□ Delete records older than: 1 day(s)
Table action
● insert new row
○ update/select
● first ○ last ○ custom ○ key/value pairs
Where:

```
Running
 Last execution: Wed May 13 09:05:08 PDT 2020         Total executions: 66
   Last trigger: Wed May 13 09:05:08 PDT 2020        Failed executions: 0
 Next execution: Wed May 13 09:05:09 PDT 2020          OPC/Tag writes: 0
  Last duration: 0.0 second(s)                               DB writes: 132
   Avg duration: 0.0 second(s)               OPC/Tag write failures: 0
```

This configuration will allow for tag1's value to be stored into a database table called **'group_table'** every 1 second to a column called 'tag1.'

We can see this working now through the Database Query Browser.



Database Query Browser

```
SELECT * FROM group_table order by t_Stamp desc
```

☑ Limit SELECT to: 1000 rows

Resultset 1

group_tabl...	tag1	t_stamp
24	16	2020-05-13 09:04:26
23	16	2020-05-13 09:04:25
22	16	2020-05-13 09:04:24
21	16	2020-05-13 09:04:23
20	16	2020-05-13 09:04:22
19	16	2020-05-13 09:04:21
18	16	2020-05-13 09:04:20
17	16	2020-05-13 09:04:19
16	16	2020-05-13 09:04:18
15	16	2020-05-13 09:04:17
14	16	2020-05-13 09:04:16
13	16	2020-05-13 09:04:15

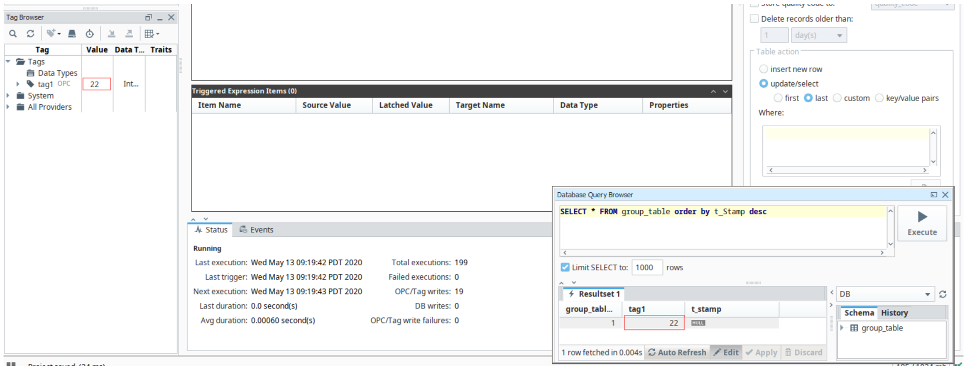
24 rows fetched in 0.001s

DB to OPC

DB to OPC Update Mode allows you to write data from your Database to an OPC tag. This can be done by configuring the following:

1. Create a Standard Transaction Group and from your Tag Browser, drag a single tag into your Transaction Group's Basic OPC/Group Items section. For this example, the tag will be called 'tag1.'
2. Set the mode of the Transaction Group to **DB to OPC** and set the Mode for tag1 to **DB to OPC**.
3. Set the Transaction Groups Table Action to **'update/select'** and check the **'last'** option. What this will do is ensure that we do not have a new value inserted to the database. What we will have instead is a single row of data in the table group_table where the value of the 'tag1' column will control tag1's OPC value.

   Testing these settings, we can see that when the value in the Database Query Browser for column 'tag1' is 22, the value for 'tag1' is also 22. When we change the value on column 'tag1' to 29, we see tag1's value change to 29 as well.



Tag Browser

Tag	Value	Data T...	Traits
Tags			
Data Types			
tag1 OPC	22	Int...	
System			
All Providers			

Triggered Expression Items (0)

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties

Running

Last execution: Wed May 13 09:19:42 PDT 2020 Total executions: 199
Last trigger: Wed May 13 09:19:42 PDT 2020 Failed executions: 0
Next execution: Wed May 13 09:19:43 PDT 2020 OPC/Tag writes: 19
Last duration: 0.0 second(s) DB writes: 0
Avg duration: 0.00060 second(s) OPC/Tag write failures: 0

□ Delete records older than: 1 day(s)

Table action
○ insert new row
● update/select
○ first ● last ○ custom ○ key/value pairs
Where:

Database Query Browser

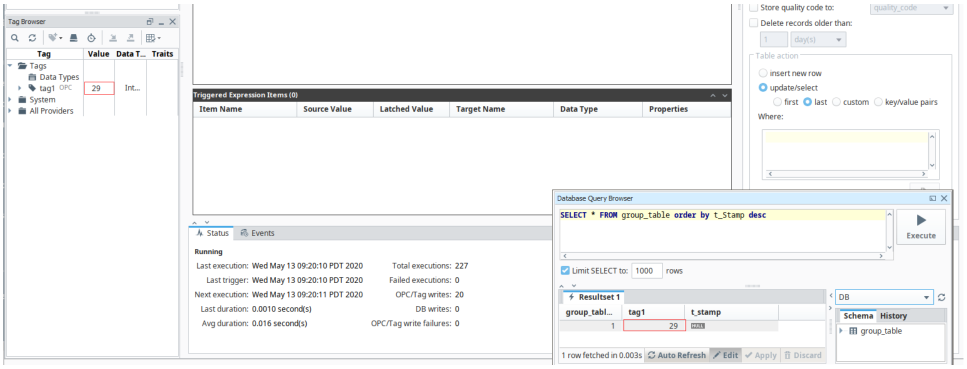
```
SELECT * FROM group_table order by t_Stamp desc
```

☑ Limit SELECT to: 1000 rows

Resultset 1

group_tabl...	tag1	t_stamp
1	22	NULL

1 row fetched in 0.004s



Tag Browser

Tag	Value	Data T...	Traits
Tags			
Data Types			
tag1 OPC	29	Int...	
System			
All Providers			

Triggered Expression Items (0)

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties

Running

Last execution: Wed May 13 09:20:10 PDT 2020 Total executions: 227
Last trigger: Wed May 13 09:20:10 PDT 2020 Failed executions: 0
Next execution: Wed May 13 09:20:11 PDT 2020 OPC/Tag writes: 20
Last duration: 0.0010 second(s) DB writes: 0
Avg duration: 0.016 second(s) OPC/Tag write failures: 0

□ Store quality code to: quality_code
□ Delete records older than: 1 day(s)

Table action
○ insert new row
● update/select
○ first ● last ○ custom ○ key/value pairs
Where:

Database Query Browser

```
SELECT * FROM group_table order by t_Stamp desc
```

☑ Limit SELECT to: 1000 rows

Resultset 1

group_tabl...	tag1	t_stamp
1	29	NULL

1 row fetched in 0.003s

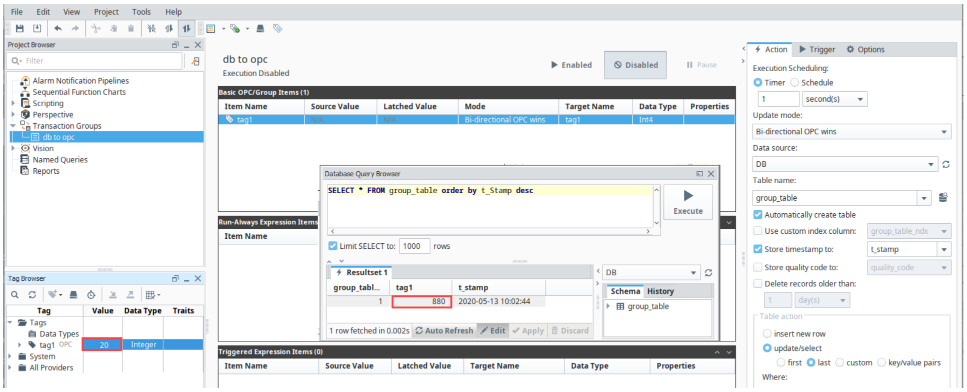
Bi-directional OPC Wins

Bi-directional OPC wins means that Ignition will Read and Write to both the database and OPC Server. However, on initial group start, if the OPC and database values are different, the OPC value will win and the Transaction Group will write OPC values to the database.

1. Create a Standard Transaction Group and from your Tag Browser, drag a single tag into your Transaction Group's Basic OPC/Group Items section. For this example, the tag will be called 'tag1.'
2. Set the mode of the Transaction Group to **'Bi-directional OPC wins'** and set the Mode for tag1 to **'Bi-directional OPC wins.'**
3. Set the Transaction Groups Table Action to **'update/select'** and check the **'last'** option. What this will do is ensure that we do not have a new value inserted to the database. What we will have instead is a single row of data in the table group_table where the value of the 'tag1' column will control tag1's OPC value and similarly, 'tag1's OPC value will control the value of the 'tag1' column database side.

   What you will have at this point is a bi-directionally controlled Transaction Group where any change to tag1's value will be reflected on the database and any change database side for the 'tag1' column value will be reflected on your 'tag1' tag.

   In the event that the OPC and database values do not match on Transaction Group start, the OPC value will win and it will be written to the database. For instance, if Transaction Group is disabled and the 'tag1' value is 20 and the 'tag1' column value is 880, the Update Mode **'Bi-directional OPC wins'** means the 'tag1' column value will be set to 20 when the Transaction Group starts.



db to opc
Execution Disabled

Basic OPC/Group Items (1)

Item Name	Source Value	Latched Value	Mode	Target Name	Data Type	Properties
tag1	N/A	N/A	Bi-directional OPC wins	tag1	Int4	

Run-Always Expression Items

Item Name

Triggered Expression Items (0)

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties

Database Query Browser

```
SELECT * FROM group_table order by t_Stamp desc
```

☑ Limit SELECT to: 1000 rows

Resultset 1

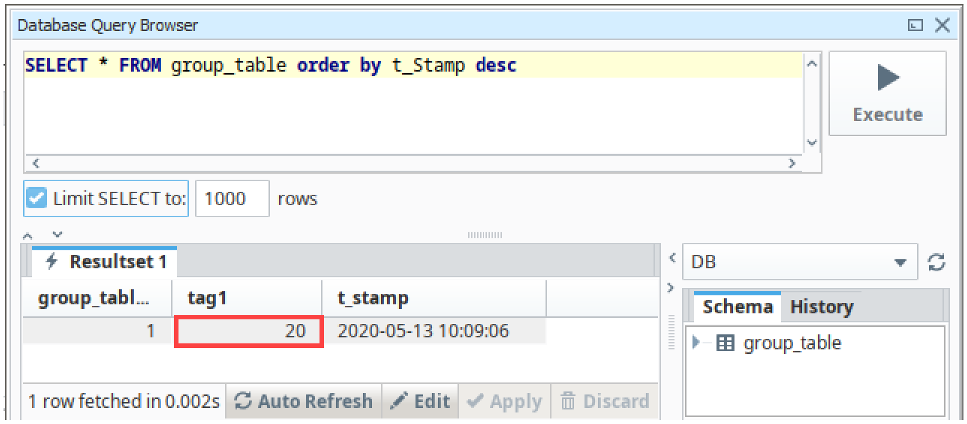
group_tabl...	tag1	t_stamp
1	880	2020-05-13 10:02:44

1 row fetched in 0.002s | Auto Refresh | Edit | Apply | Discard

DB

Tag	Value	Data Type	Traits
Tags			
Data Types			
tag1 OPC	20	Integer	
System			
All Providers			

Execution Scheduling:
● Timer ○ Schedule
1 second(s)
Update mode: Bi-directional OPC wins
Data source: DB
Table name: group_table
☑ Automatically create table
☐ Use custom index column: group_table_ndx
☑ Store timestamp to: t_stamp
☐ Store quality code to: quality_code
☐ Delete records older than: 1 day(s)
Table action
○ insert new row
● update/select
○ first ● last ○ custom ○ key/value pairs
Where:



Database Query Browser

```
SELECT * FROM group_table order by t_Stamp desc
```

☑ Limit SELECT to: 1000 rows

Resultset 1

group_tabl...	tag1	t_stamp
1	20	2020-05-13 10:09:06

1 row fetched in 0.002s | Auto Refresh | Edit | Apply | Discard

DB

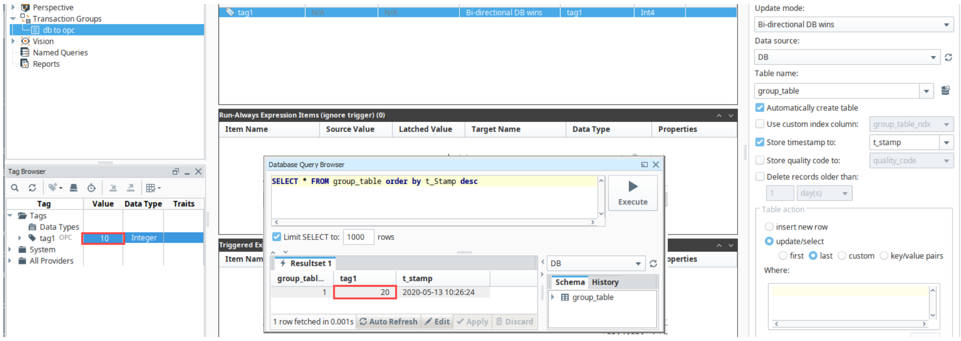
Bi-directional DB Wins

Bi-directional DB wins means that Ignition will Read and Write to both the database and OPC Server. However, on initial group start, if the OPC and database values are different, the database value will win and the Transaction Group will write database data to your OPC data points.

1. Create a Standard Transaction Group and from your Tag Browser, drag a single tag into your Transaction Group's Basic OPC/Group Items section. For this example, the tag will be called 'tag1.'
2. Set the mode of the Transaction Group to **'Bi-directional DB wins'** and set the Mode for 'tag1' to **'Bi-directional DB wins.'**
3. Set the Transaction Groups Table Action to **'update/select'** and check the **'last'** option. What this will do is ensure that we do not have a new value inserted to the database. What we will have instead is a single row of data in the table group_table where the value of the 'tag1' column will control tag1's OPC value and similarly, tag1's OPC value will control the value of the 'tag1' column database side.

   What you will have at this point is a bi-directionally controlled Transaction Group where any change to tag1's value will be reflected on the database and any change database side for the 'tag1' column value will be reflected on your 'tag1' tag.

   In the event that the OPC and database values do not match on Transaction Group start, the database value will win and it will be written to the OPC data point. For instance, when the Transaction Group is disabled and 'tag1' value is 10, and the 'tag1' column value is 20, the Update Mode being **'Bi-directional DB wins'** means the tag1 tag value will be set to 20 when the Transaction Group starts.



Tag	Value	Data Type	Traits
Tags			
Data Types			
tag1 OPC	10	Integer	
System			
All Providers			

tag1	N/A	N/A	Bi-directional DB wins	tag1	Int4

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties

```
SELECT * FROM group_table order by t_Stamp desc
```

☑ Limit SELECT to: 1000 rows

Resultset 1

group_tabl...	tag1	t_stamp
1	20	2020-05-13 10:26:24

1 row fetched in 0.001s Auto Refresh Edit Apply Discard

DB

Update mode: Bi-directional DB wins

Data source: DB

Table name: group_table

☑ Automatically create table

□ Use custom index column: group_table_ndx

☑ Store timestamp to: t_stamp

□ Store quality code to: quality_code

□ Delete records older than: 1 day(s)

Table action

○ insert new row

● update/select

○ first ● last ○ custom ○ key/value pairs

Where:

Related Topics ...

- Understanding Transaction Groups
- Types of Groups