

1. Tag Historian .....	2
1.1 Tag History Gateway Settings .....	5
1.2 Configuring Tag History .....	12
1.3 Data Partitioning and Pruning .....	18
1.4 Custom Tag History Aggregates .....	21
1.5 Tag History Splitter .....	24
2. SQL Bridge (Transaction Groups) .....	28
2.1 Understanding Transaction Groups .....	32
2.1.1 Types of Groups .....	42
2.1.2 Item Types .....	49
2.1.3 Hour and Event Meters .....	57
2.2 Transaction Group Examples .....	63
2.2.1 Block Group .....	66
2.2.2 Recipe Group .....	73
2.2.3 Update or Insert Group .....	77
2.2.4 Trigger Options .....	80
2.2.5 Transaction Group Update Modes .....	84
2.2.6 OPC to OPC Transaction Group .....	89

# Tag Historian

## Overview

Ignition has two main approaches to recording historical data:

- **Tag Historian Module:** Individual or groups of tags can be configured to record history based on scan class execution.
- **Transaction Groups:** Groups of OPC items that are recorded on an execution cycle. More information on Transaction Groups can be found in the [SQL Bridge \(Transaction Groups\)](#) section.

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

...

- Overview
- Historian Tables
- Tag Historian Querying
- Vision Historian Charts
- Store and Forward
- Other Methods of Storing Historical Data
- Data storage

## Historian Tables

With the [Vision Table component](#) or the [Perspective - Table component](#) you can quickly make custom tables that display historical data. You can customize your table to bring back the most recent history, a specific date range, fixed sample size, and interval sample size. In Vision, you can drag and drop history-enabled tags onto a table component to display historical values.

t_stamp	Tag1	Tag2
Jul 11, 2016 11:57 AM	10	25
Jul 11, 2016 11:58 AM	10	25
Jul 11, 2016 11:58 AM	10	25
Jul 11, 2016 11:58 AM	10	25
Jul 11, 2016 11:58 AM	10	25
Jul 11, 2016 11:58 AM	10	25
Jul 11, 2016 11:58 AM	10	25
Jul 11, 2016 11:59 AM	10	25
Jul 11, 2016 11:59 AM	10	25
Jul 11, 2016 11:59 AM	10	25
Jul 11, 2016 11:59 AM	10	25
Jul 11, 2016 11:59 AM	10	25
Jul 11, 2016 11:59 AM	10	25
Jul 11, 2016 12:00 PM	10	25
Jul 11, 2016 12:00 PM	10	25
Jul 11, 2016 12:00 PM	10	25

## 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.

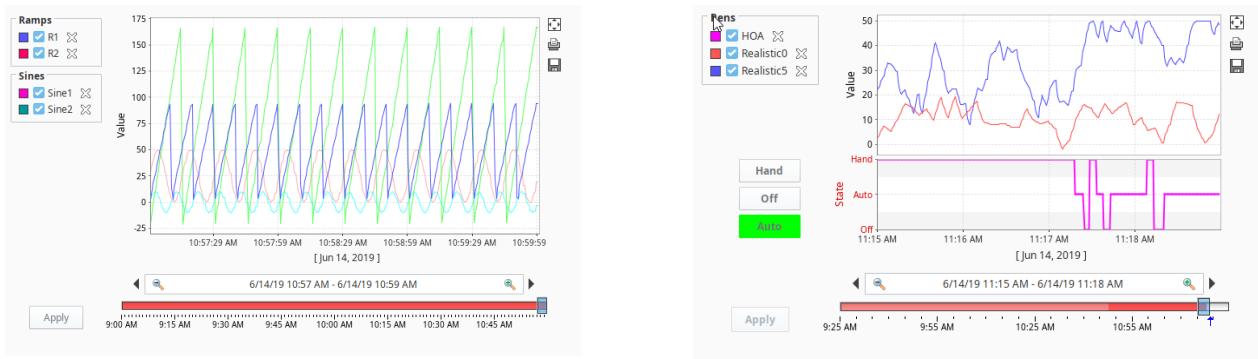
For more information, see [Tag History Bindings in Perspective](#) or [Tag History Bindings in Vision](#).

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](#).

## Vision Historian Charts

In the Vision module, you can use the [Easy Chart component](#) to make powerful timeseries charts from Tag Historian data. 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.

To see all the chart options and features for the Vision module's Easy Chart, refer to the section on [Using the Vision Easy Chart](#).



## 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.

## Other Methods of Storing Historical Data

The [SQL Bridge \(Transaction Groups\)](#) Module performs a variety of tasks to store data historically. In their simplest form, Transaction Groups read values from the OPC addresses and store them into a SQL database. There are four types of Transaction Groups; Standard, Block, Historical, and Stored Procedures.

- **Standard Group** is the most flexible, and can also write database values to OPC addresses or synchronize data changes between both the database and PLC. This allows you to create true realtime values tables in a database, and push values to a PLC.
- **Historical Group** can quickly store data from the plant floor into any kind of SQL database.
- **Block Group** transfers large amounts of data very efficiently.
- **Stored Procedures** uses PLC data as inputs and outputs from your existing stored procedures.

## 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 datatype 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.

In This Section ...

# Tag History Gateway Settings

## Configuring Tag History Settings

Tag History storage is easy to set up quickly, but there are also some settings that can be adjusted to allow for differences in database storage space and performance needs.

### On this page

...

- Configuring Tag History Settings
  - Table Partitioning
  - Data Compression
- Datasource History Providers
  - Editing Datasource History Providers
  - OPC-HDA Provider
  - Internal History Provider
  - Remote History Provider
  - Tag History Splitter

### Table Partitioning

Ignition has the ability to automatically break up data into different tables of fixed duration. This can help make data maintenance easier by preventing tables from becoming too large. Tables can easily be deleted in order to prune old data, and the database is able to better optimize access to frequently retrieved rows. The built-in partitioning feature can be used with any database.

It is important to note the difference between this feature and any partitioning options that the database might provide. Most modern databases offer their own faculties for defining "partitions", offering similar and greater benefits. While Ignition cannot use these features directly, advanced users may choose to apply these features on top of what Ignition currently offers.

### Data Compression

Ignition does not perform any binary compression on the data. That is, values are stored directly in standard database tables. However, to reduce the number of values stored, Ignition offers two different algorithms for pre-compressing the data (trimming unnecessary values). The two modes correspond to the value mode property of the Tag. The value mode (**Discrete** or **Analog**) dictates the type of value that the Tag represents, affects how the deadband is applied to values, and how interpolation is performed when querying.

- **Discrete**

The value uses a simple deadband and is only stored when a new value is + /- the deadband value away from the previously stored value.

- **Analog**

The deadband is used to form a corridor along the trajectory of the value. A new value is only stored when it falls outside the previous corridor. When this occurs, the trajectory is recalculated and a new corridor is formed.

Typically, Discrete is used for boolean or integers that represent state, and Analog is used for floats or integers that change more often (which is why you want to perform compression). While advanced users can change the table according to their database to be more efficient (for example, using a compressed engine), Ignition does not perform binary compression or encrypt the data in any way. See Deadband Style, in [Tag Properties Table](#) for more information about the difference between Discrete and Analog values.

### Datasource History Providers

Datasource History Providers can not be created or deleted, but are instead tied to a database connection. They are automatically added when connecting to a new database and removed after the database connection is removed. It comes pre-configured to partition every month, but the provider can be edited to change its behavior.

There are two other major options to configure on the provider: pre-processed partitions and data pruning. With pre-processed partitions, the data that is stored is summarized and then placed into another table in the database. While this takes up more space in the database, it can improve query speed by reducing the amount of data points that must be loaded. Data pruning will automatically remove old data from your system after it reaches an age that you set. It will only remove whole tables though. If each partitioned table represents a month and the pruning system removes data that is three months old, it will wait until all the data in the oldest table is three months old before pruning it.

### Editing Datasource History Providers

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 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	Pre-processed partitions will use more space in the database, but can improve query speed by summarizing data, reducing the amount that must be loaded.
Pre-processed Window Size (seconds)	When pre-processing is turned on, the data will be summarized into blocks of this 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.  <div style="border: 1px solid #f0c080; padding: 5px; margin-left: 20px;">  Data pruning works by deleting old partitions. Therefore, data will only be removed when a partition has no data younger than the prune age.         </div>
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 scan class 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 scan class rate used to determine when values are stale. If scan class execution is not recorded within this amount of time, values will be considered bad on query.

## OPC-HDA Provider

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.



This requires the [OPC COM](#) to be installed.

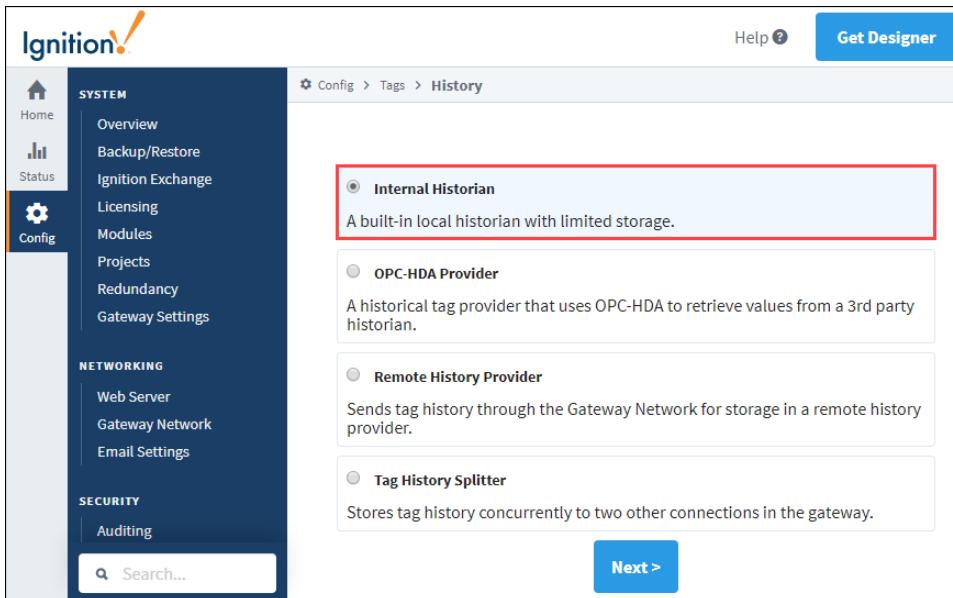
The following feature is new in Ignition version **8.0.4**  
[Click here](#) to check out the other new features

## Internal History Provider

As of version 8.0.4, the Edge Historian, an internal history provider, is available on standard Ignition Gateways.

To set up an Edge history provider, do the steps the follow:

1. Go to the **Config** section of the Gateway Webpage and select **Tags > History**.
2. Click **Create New Historical Tag Provider**.
3. Select the **Internal Historian** radio button and click **Next**.



4. Fill in the properties in the table.

Main	
Provider Name	Name of the Edge 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 Tag Historian Module License)	
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 (Requires EAM Module Licensing)

Remote Sync Enabled	Allows you to turn <a href="#">Tag History Synchronization</a> on or off. Default is false.
Remote Gateway Name	The Gateway to target for remote synchronization. Must have the <a href="#">Tag Historian</a> module installed, and allow remote storage. The Ignition Gateway's <a href="#">security settings</a> 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: <ul style="list-style-type: none"> <li>• 9:00-15:00</li> <li>• 9pm-5am</li> <li>• 20.30-04.30</li> </ul>

5. After filling in the properties in the table as desired, click **Create New Historical Tag Provider**.

Once an Edge History Provider is set up, you can select it as the storage provider for your tags. For example, in the following image, and Edge History Provider named "Internal Historian Test" is selected for storing history on a memory tag.

Tag Editor

Mem Tag 1

default

Properties

Meta Data Properties	
Tooltip	
Documentation	
Security	
Access Rights	Read/Write
Scripting	
Tag Event Scripts	0 event scripts
Alarms	
Alarms	No alarms
Alarm Eval Enabled	true
History	
History Enabled	true
Storage Provider	Internal Historian Test
Deadband Style	<None>
Deadband Mode	Internal Historian Test
Historical Deadband	vm_db
Sample Mode	MySQL
Min Time Between Samples	1

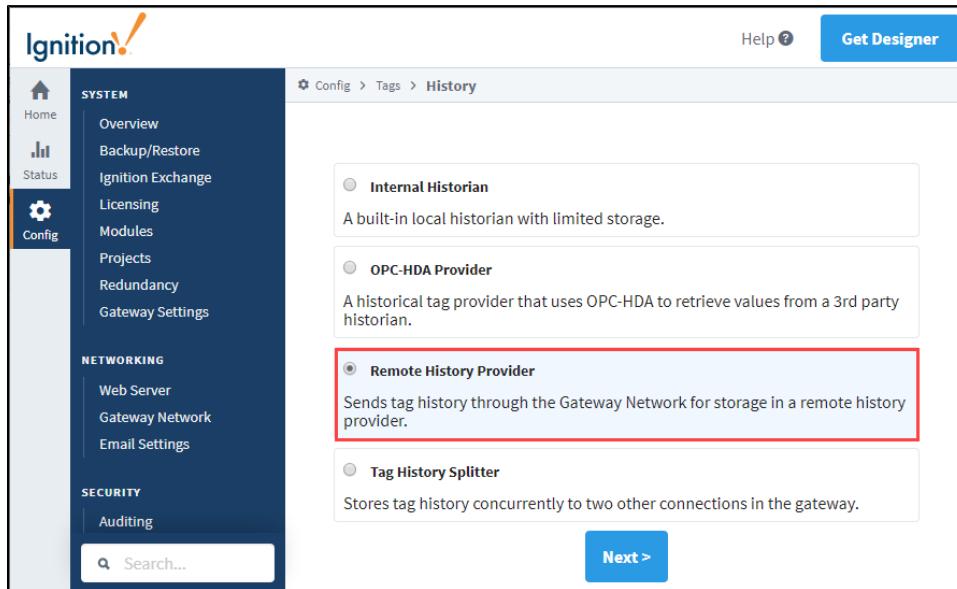
## Remote History Provider

A Remote History Provider is a link to a historical provider on another gateway. Since it is grabbing historical tag data from another provider, its only configuration is to ensure it is pointed at the correct tag provider. 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.

To set up a Remote History Provider, do the steps that follow:

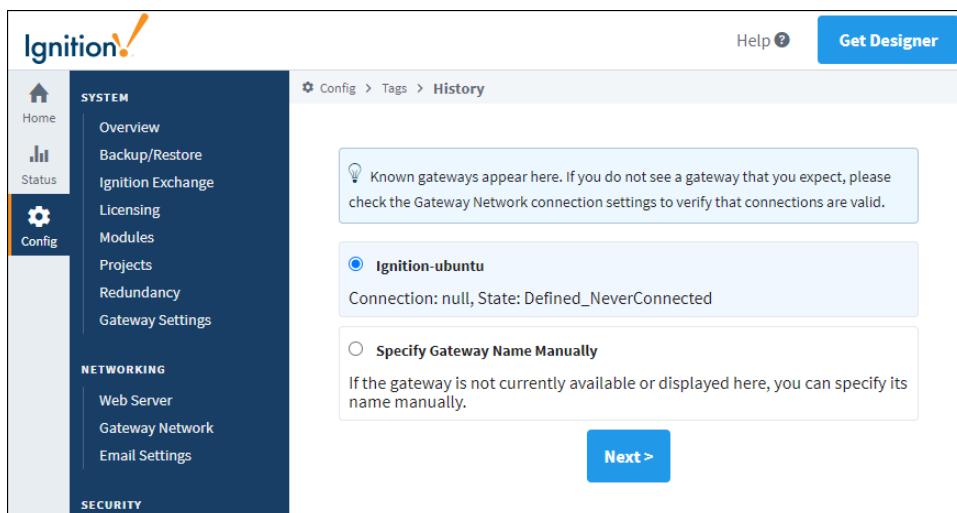
1. Go to the **Config** section of the Gateway Webpage and select **Tags > History**.

2. Select the **Remote History Provider** radio button.



The screenshot shows the Ignition configuration interface with the 'Config' tab selected. The left sidebar has 'Config' highlighted. The main content area is titled 'Config > Tags > History'. It lists four provider options: 'Internal Historian', 'OPC-HDA Provider', 'Remote History Provider' (which is selected and highlighted with a red box), and 'Tag History Splitter'. A 'Next >' button is at the bottom right.

3. A list of known Gateways appears. If the Gateway is not currently available or displayed here, you can specify its name manually.



The screenshot shows the Ignition configuration interface with the 'Config' tab selected. The left sidebar has 'Config' highlighted. The main content area is titled 'Config > Tags > History'. It displays a note about known gateways and a list of gateways. 'Ignition-ubuntu' is selected and highlighted with a blue box. A 'Specify Gateway Name Manually' option is also shown. A 'Next >' button is at the bottom right.

4. Select a Gateway and click **Next**.

5. Fill in the properties in the table.

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.
<b>Storage</b>	
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.
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. 0=unlimited

6. After filling in the properties in the table, click **Create New Historical Tag Provider**.

## Tag History Splitter

This provider combines two separate providers into a single new provider. 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. Learn more about setting up the [Tag History Splitter](#) here.

### Related Topics ...

- [Configuring Tag History](#)

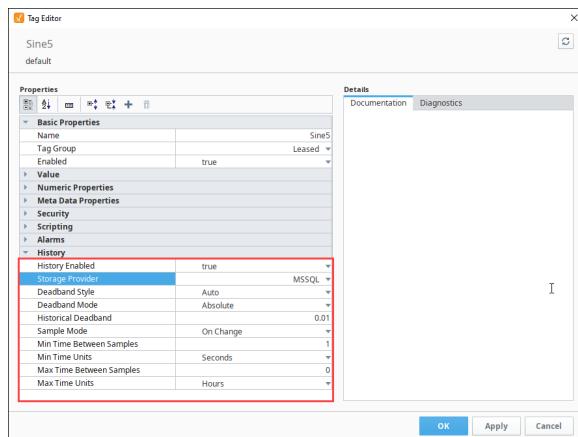
# Configuring Tag History

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 datatype directly to a table in the SQL database, along with its quality and a millisecond resolution time stamp. 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.

## Tag Configuration

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. Complete information on the History properties (and all properties in the Tag Editor), can be found on the [Tag Properties Table](#).



## Sample Mode

The Sample Mode setting determines how often a historical record should be collected.

- **On Change** - Collects a record whenever the value on the Tag changes.
- **Periodic** - Collects a record based on the **Sample Rate** and **Sample Rate Units** properties.
- **Tag Group** - Collects a record based on the Tag Group specified under the **Historical Tag Group** property.

## Historical Tag Group

Historical Tag Group setting shows up with Sample Mode is set to Tag Group. Historical Tag Group setting determines how often to record the value on the Tag. 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: 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 scan class 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.

## On this page

...

- [Tag Configuration](#)
  - [Sample Mode](#)
  - [Max and Min Time Between Samples](#)
- [Deadband and Analog Compression](#)
- [Seeded Values](#)
- [Raw Data Queries](#)
- [Log Tag History Data](#)
- [Setting a UDT to Log History Data](#)



## Configuring Tag History

[Watch the Video](#)

# 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

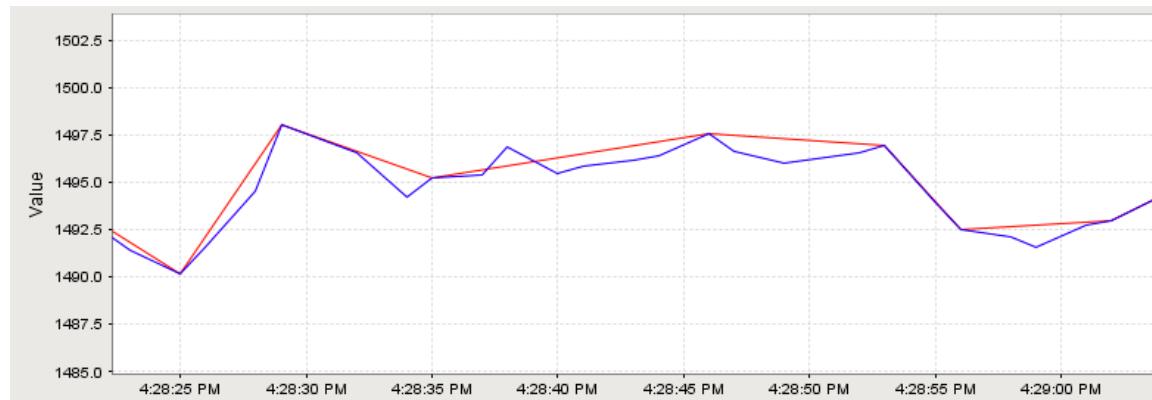
- **Storage** - The deadband will be applied directly to the value. That is, a new value ( $V_1$ ) will only be stored when:  $|V_1 - V_0| \geq \text{Deadband}$ .
- **Interpolation** - The value will not be interpolated. The value returned will be the previous known value, up until the point at which the next value was recorded.

## Analog

- **Storage** - The deadband is used to form a corridor along the trajectory of the value. A new value is only stored when it falls outside the previous corridor. When this occurs, the trajectory is recalculated, and a new corridor formed. See below for an example.
- **Interpolation** - The value will be interpolated linearly between the last value and the next value. For example, if the value at Time<sub>0</sub> was 1, and the value at Time<sub>2</sub> is 3, selecting Time<sub>1</sub> will return 2.

## 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.



In this image, 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 Tag 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.

By looking at one specific sequence, we can see how the algorithm works:



The sequence starts with the second stored compressed value on the chart.

1. A value is stored. No further action is taken.
2. The next value arrives. A line is made through the value, with the size of the specified deadband value. A line is projected from the last stored value to the upper (line U1), and lower (line L1), bounds of this new value line. This establishes the initial corridor.
3. A new value arrives. The same procedure is taken, and new lines are created. However, only lines that are more restrictive than the previous are used. In this case, that means only line U2, the new upper line.
4. Another value arrives, causing a new lower line (L3) to be used.
5. Finally, a value arrives that falls outside of our corridor. The last received value (value 4) is stored, and the process is started again from that point.

## 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 the tag is storing history with an Analog Value Mode, or "Prevent Interpolation" is enabled on the calling query, then these seeded values will not be included.

### 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.

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.

## 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:

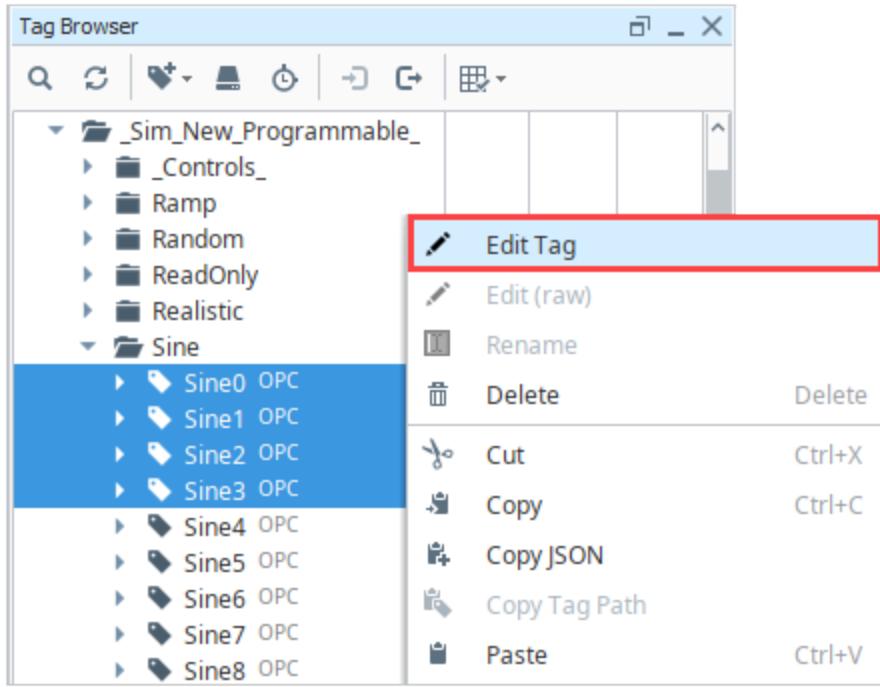
- Set the Sample Size on Vision Tag History bindings to **On Change**
- Setting the `returnSize` parameter on `system.tag.queryTagHistory` or `system.tag.queryTagCalculations` to `-1`
- Setting the Query Mode on Perspective Tag History bindings to **AsStored**

## Log Tag History Data

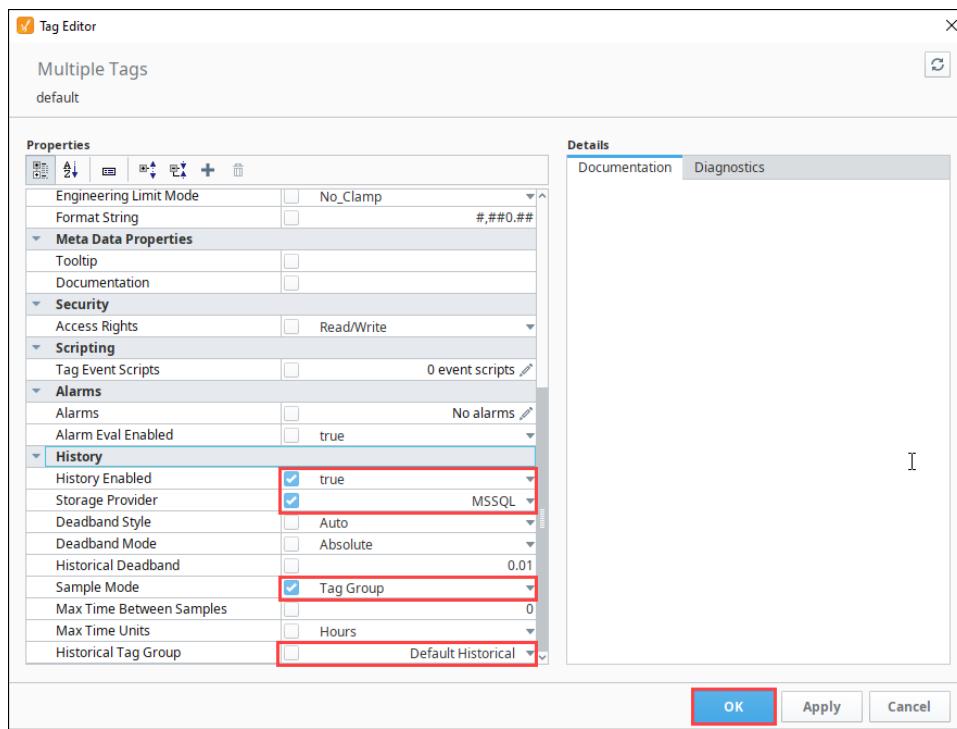
**Note:** Dataset type tags are not supported by the Tag History system.

Do the following steps to log history data for your Tags:

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. Here, you can edit the Tag and change the name, data type, scaling options, metadata, permissions, history, and alarming.



3. Scroll down to the **History** section of the Tag Editor. Select the **History Enabled** check box.
4. Choose a database (for example, MySQL) from the **Storage Provider** dropdown.
5. Set the Sample Mode to **Tag Group**.
6. Set the Historical Tag Group to **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.



Path	Type	Value	Unit	Min	Max	Scale	Source
ReadOnly	Double	-94.78	Double	-94.78	-94.78	Double	OPC
Realistic	Double	9.3	Double	9.3	9.3	Double	OPC
Sine	Double	0.92	Double	0.92	0.92	Double	OPC
Sine0	Double	38.12	Double	38.12	38.12	Double	OPC
Sine1	Double	-29.42	Double	-29.42	-29.42	Double	OPC
Sine2	Double	-94.78	Double	-94.78	-94.78	Double	OPC
Sine3	Double	-94.78	Double	-94.78	-94.78	Double	OPC
Sine4	Double	-94.78	Double	-94.78	-94.78	Double	OPC
Sine5	Double	-94.78	Double	-94.78	-94.78	Double	OPC

If you were to look in your database, you can see all the tables and data Ignition has created for you.

## Setting a UDT to Log History Data

You can set a [UDT](#) to log history data, then all the instances of that UDT will log data without having to edit the individual instances.

1. In the Tag browser, right-click on the UDT (for example, a Motor UDT) and select **Edit Tag**.  
The Tag Editor is displayed.
  2. In Tag Editor, click a Tag (for example, the AMPS Tag). Scroll down to the **History** section.

Set the following properties in the Tag Editor:  
**History Enabled:** true  
**Storage Provider:** MSSQL  
**Sample Mode:** Tag Group  
**Historical Tag Group:** Default Historical



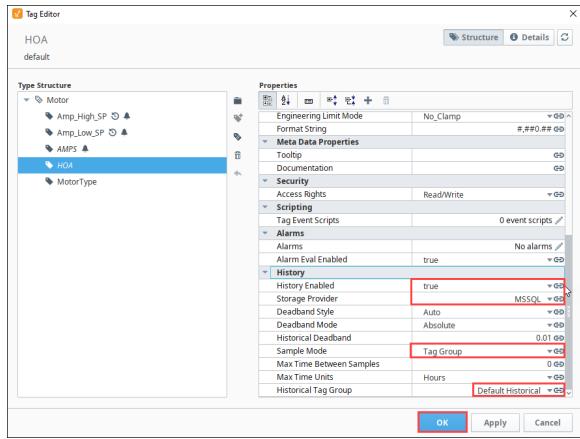
## Add History to Tags in UDT

## Watch the Video

Properties

Engineering Limit Mode	No_Clamp
Format String	##.##.##
<b>Meta Data Properties</b>	
Documentation	
Security	ReadWrite
Scripting	0 event scripts
<b>Alarms</b>	
Alarms	1 configured alarm(s)
Alarm Eval Enabled	true
<b>History</b>	
History Enabled	true
Storage Provider	MSSQL
Deadband Style	Auto
Deadband Mode	Absolute
Historical Deadband	0.01
Sample Mode	Tag Group
Max Time Between Samples	0.06
Max Time Units	Hours
Historical Tag Group	Default Historical

4. Click **Apply**.
  5. Next, select the HOA Tag.
  6. Set the following properties in the Tag Editor:
    - History Enabled: **true**
    - Storage Provider: **MSSQL**
    - Sample Mode: **Tag Group**
    - Historical Tag Group: **Default Historical**
  7. Click **OK** to save the changes to the UDT. Now every motor instance automatically starts logging data.



# Data Partitioning and Pruning

Tag Historian will partition data into separate tables according to the time setting so that one table doesn't grow indefinitely, and then will delete old data to ensure the system is maintained for query performance. By default, partitioning is enabled to improve query performance. Tag Historian partitions and breaks up the data into separate tables 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. The data prune feature will delete partitions with data older than a specific age/time.

## On this page

...

- Partition and Prune Data
- History Table Timestamps



## Data Partitioning and Pruning

[Watch the Video](#)

## Partition and Prune Data

1. Go to the **Config** tab of the Gateway.
2. Select **Tags > History** from the menu on the left.

The **Historical Tag Providers** page is displayed. You can see the Databases that have Enabled tag history on and their Status shows as Running.

The screenshot shows a table of historical tag providers. The columns are: Provider Name, Enabled, Type, Description, and Status. There are two rows: one for 'MySQL' (Datasource History Provider, Enabled, Type: Datasource History Provider, Status: Running, with an 'edit' button highlighted with a red box) and one for 'New History Provider' (Tag History Splitter, Enabled, Type: Tag History Splitter, Status: Running, with 'edit' and 'delete' buttons). Below the table is a link to 'Create new Historical Tag Provider...' and a note: 'Note: For details about a provider's status, see the Tag Providers Status page.'

Provider Name	Enabled	Type	Description	Status	
MySQL	true	Datasource History Provider		Running	<a href="#">edit</a>
New History Provider	true	Tag History Splitter		Running	<a href="#">delete</a> <a href="#">edit</a>

→ [Create new Historical Tag Provider...](#)

**Note:** For details about a provider's status, see the [Tag Providers Status page](#).

3. Click on **edit** at the far right of the provider you want to update.
4. Once you've made changes, click **Save Changes** at the bottom of the screen.

The following table describes all the settings available for Tag History:

Main	
Provider Name	Name of the Tag History Provider, for example, MySQL.
Enabled	By default, the check box is selected (enabled) meaning the provider is turned on and accepts tag history data.
Description	Description of the Tag History Provider (optional).
Data Partitioning	
Enable Partitioning	The built-in partitioning system breaks up data into separate tables of a specified time frame. This can improve performance and make certain maintenance tasks easier. Default is true.
Partition Length	The size of each partition, the default is one table per month. Many systems whose primary goal is to show only recent data might use smaller values, such as a week, or even a day. Default is 1.
Partition Units	Unit of time for the partition length. Options are: Milliseconds, Seconds, Minutes, Hours, Days, Weeks, Months, and Years. Default is Months.
Enable Pre-processed Partitions	Pre-processed partitions will use more space in the database, but can improve query speed by summarizing data, reducing the amount that must be loaded. Default is false.
Pre-processed Window Size (seconds)	When pre-processing is turned on, the data will be summarized into blocks of this size. Default is 60.
Data Pruning	
Enable Data Pruning	Partitions with data older than a specific age are deleted and if the data is not archived, the data is then lost. Default is false. <div style="border: 1px solid #f0e68c; padding: 5px; margin-top: 10px;">  Data pruning works by deleting old partitions. Therefore, data will only be removed when a partition has no data younger than the prune age.           </div>
Prune Age	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. Default is 1.
Prune Age Units	Unit of time for the prune age. Options are: Milliseconds, Seconds, Minutes, Hours, Days, Weeks, Months, and Years. Default is Years.
Show Advanced Properties	Select this option to display the Advanced properties below:
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. Default is true.
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. Default is 2.

## History Table Timestamps

If you've looked behind the scenes of SQLTags Historian, you've probably noticed the timestamps are not stored as standard SQL timestamps. They are stored in a variant of Unix time, or the number of milliseconds since January 1, 1970 00:00:00. The time may come when you need to convert that timestamp to a more human-readable format. The following describes how to do it in MySQL and MSSQL.

Both examples below assume the partition table is named 'sqlt\_data\_1\_2016\_08'.

### MySQL

It's pretty easy to deal with Unix timestamp in MySQL because they have a built-in function for doing so. The `FROM_UNIXTIME()` function will take in a Unix timestamp and return the current timestamp.

Usage:

```
SELECT FROM_UNIXTIME(t_stamp/1000) FROM sqlt_data_1_2016_08
```

### **MSSQL**

In Microsoft SQL Server, it's a little more verbose. We use the DATEADD() function to figure out the timestamp.  
Usage:

```
SELECT DATEADD(s,t_stamp/1000,'1970-01-01 00:00:00') FROM sqlt_data_1_2016_08
```

# 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 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.



When calling a custom tag history aggregate, the `returnSize` argument must be set to `natural` (`returnSize = 0`). If the `returnSize` is set to `-1`, or left with its default value, the the custom aggregate will be ignored.

## On this page

...

- Python Aggregation Functions
- Description
  - Parameters
  - Return Value
- Usage
- Examples
  - Using a Shared Script
  - Creating an Aggregate Function on the Fly

## 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 block context 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):

1. The start of the window
2. 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)
3. The raw value
4. 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

- 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 shared 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 shared script, and if not, will then try to compile it as a full function.

For performance reasons, it is generally recommended to use the shared script whenever possible.

## Examples

### Using a Shared Script

This example assumes a Shared Scripts named "aggregates" contained the function listed below.

#### 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)
```

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.aggregates.
myCount'], returnSize = 0)
```

## 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 = 0)
```

# Tag History Splitter

The Tag History Module has a provider type called the Tag History Splitter. Like the Remote History Provider, it doesn't store history on its own, it relies on having other providers already set up. A Splitter provider simply logs Tag History into multiple existing History Providers.

Some users prefer to have data recorded by the Tag Historian sent to multiple databases: project specifications require redundant logging, or users at another facility would like to have a copy of the data in their local database. In cases like this, the Tag History Splitter Provider is ideal.

The Gateway that these Tags reside in must have multiple Tag History Providers configured. Should one of the providers fault, the Store and Forward system will kick in to maintain the data on the faulted connection. Since each database connection has its own Store and Forward engine, the data is always forwarded to the correct database.



## Tag History Splitter

[Watch the Video](#)

## 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.

## Set Up a Tag History Splitter

To create the additional Tag History provider, do the following steps:

1. Go to the **Config** section of the Gateway Webpage, and choose **Tags > History**.
2. Click **Create new Historical Tag Provider**.
3. Select **Tag History Splitter** and click **Next**. The New Historical Tag Provider page is displayed.

Config > Tags > History

- Internal Historian**  
A built-in local historian with limited storage.
- OPC-HDA Provider**  
A historical tag provider that uses OPC-HDA to retrieve values from a 3rd party historian.
- Remote History Provider**  
Sends tag history through the Gateway Network for storage in a remote history provider.
- Tag History Splitter**  
Stores tag history concurrently to two other connections in the gateway.

**Next >**

4. Enter a name for the **Provider Name**. From the dropdown choose a database for the **First Connection** (for the primary data) and one for the **Second Connection** (for secondary data).

Config > Tags > History

Main	
Provider Name	New History Provider
Enabled	<input checked="" type="checkbox"/> Enable this tag history provider (default: true)
Description	

Storage	
First Connection	MySQL <input type="button" value="▼"/> Choose One MySQL. However, all queries execute against the MySQL connection if the first connection is unavailable. MySQL <input type="button" value="▼"/> Choose One <input type="button" value="▼"/>
Second Connection	

Querying	
Limit First Connection Query	<input type="checkbox"/> 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. (default: false)
Limit Length	1 (default: 1)
Limit Units	Months <input type="button" value="▼"/> (default: MONTH)

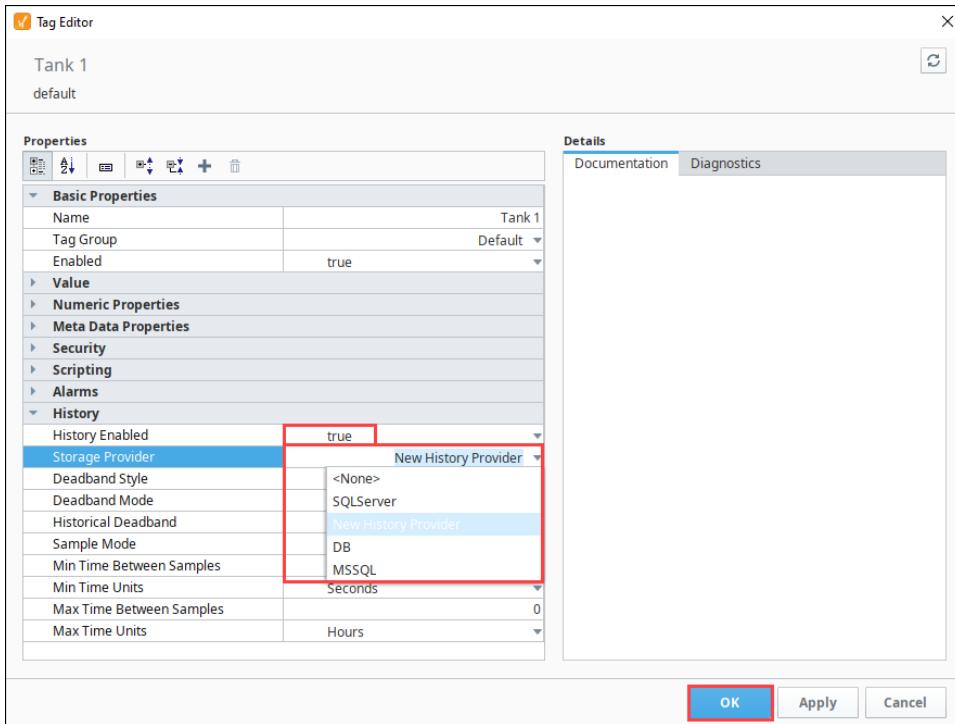
  

**Create New Historical Tag Provider**

5. Click **Create New Historical Tag Provider**.

Now the Tag History Splitter provider is created, and you can use it to log the Tag History data in the Designer. To test this, open your project in the Designer.

1. In the Tag Browser, selecting a Tag and right-click to select the **Edit Tag**  option.
2. On the Tag Editor window scroll down to the **History** section and set **History Enabled** to true.
3. In the dropdown list for Storage Provider, select the new provider.



4. Click **OK** to save the change.

# SQL Bridge (Transaction Groups)

## Overview

The SQL Bridge Module enables the creation of Transaction Groups that synchronize data between PLCs and databases. You can use Transaction Groups to easily log from PLCs to the database, move data from the database back to PLCs, and even keep the two synchronized. Drag and drop functionality makes setup of Transaction Groups quick and easy.

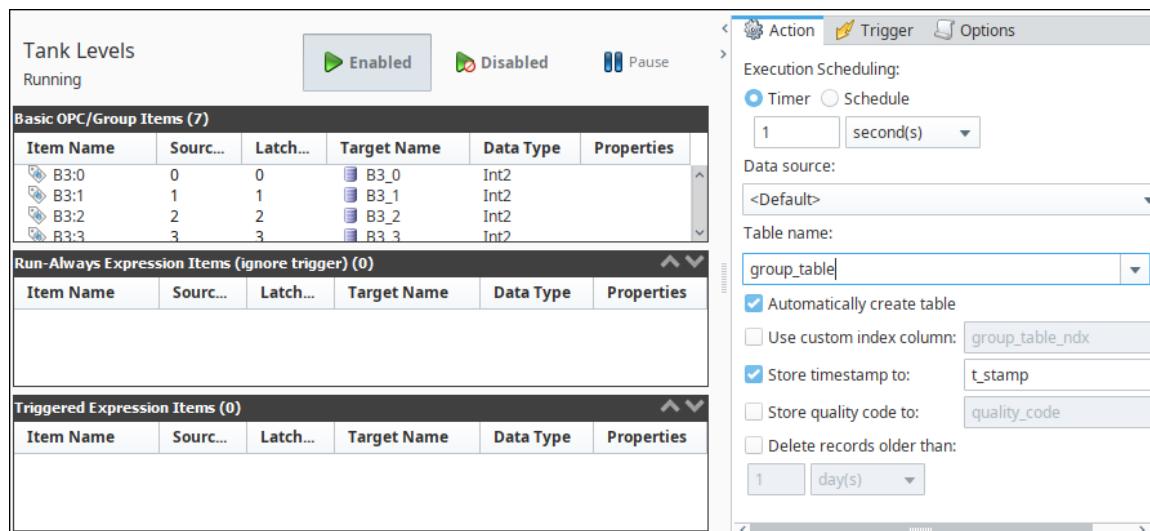
Originally conceived as an easy data storage method, Transaction Groups have become a core feature of Ignition. In their simplest form, they regularly read values from OPC addresses and store them into a SQL database. While data collection is still their primary use, they have grown in functionality over time.

To set up and use Transaction Groups, SQL knowledge is not required. Ignition can automatically create and manage the database table for each group. Prior experience writing SQL queries or creating database tables are not required to log data.

## On this page

...

- Overview
- Types of Transaction Groups
  - Historical Data Logging
  - Database and OPC Synchronization
  - Large Data Block Storage
  - Stored Procedures
  - Centralizing Data Collection



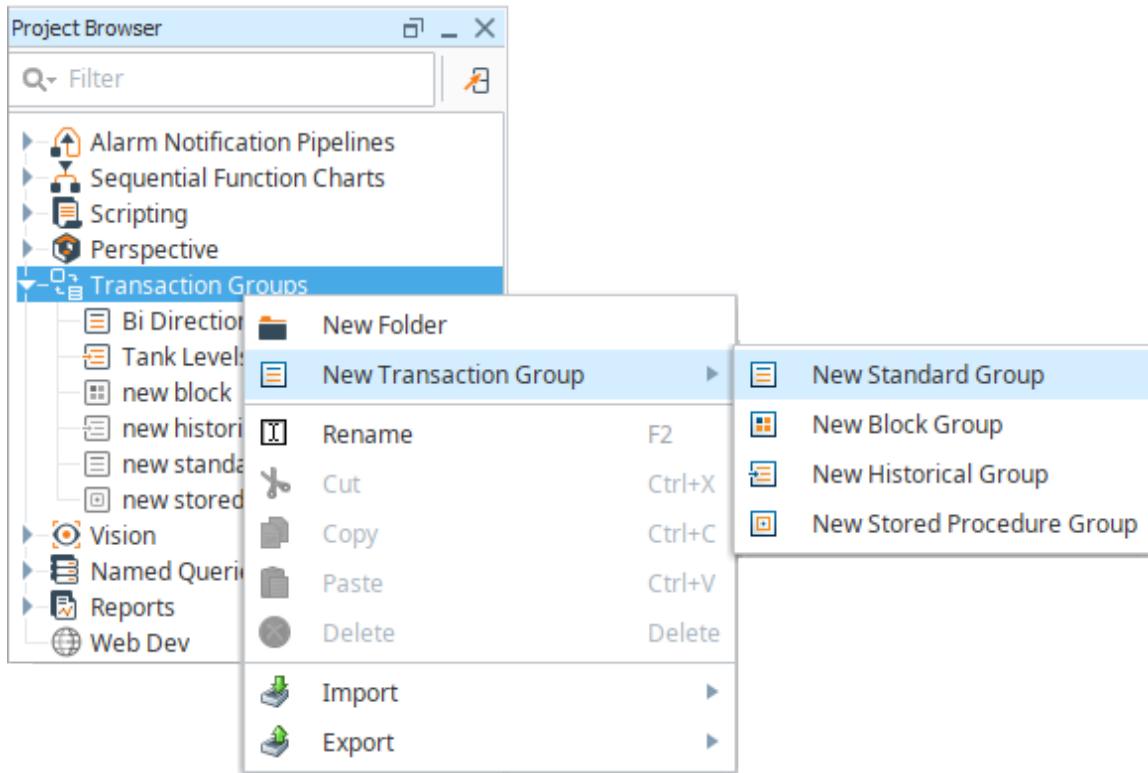
## Types of Transaction Groups

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

- **Historical Groups** - Quick and easy group that collects historical records
- **Standard Groups** - An improved version of the historical group that can reverse the flow of data, writing database values directly to Tags.
- **Block Group** - Records "blocks" of data, allowing you to record multiple values per execution in a tall format.
- **Stored Procedure Group** - Invokes a stored procedure in the database, returning the results of any OUT or INOUT parameters to Tags.

Learn more about each type of group on the [Understanding Transaction Groups](#) page.

All Transaction Groups can execute at a set rate or on a schedule. A trigger can be used to determine when the group should record. You can use Ignition's [expression language](#) in the trigger to allow complex logic to determine when logging occurs, making precision execution easy.



## Historical Data Logging

Historical Groups quickly and easily store data from the plant floor into any kind of SQL database. Items from any or all devices can be included in the same group, just drag a few Tags over and start the group running. Ignition will log the data until you tell it to stop.

The screenshot shows the Ignition interface with the following components:

- Historical Data Logging Configuration:** A window titled "new historical" showing a table of "Basic OPC/Group Items (6)". The table includes columns: Item Name, Source Va..., Latched ..., Target Name, Data Type, and Properties. Items listed include Sim\_Generic/Ramp/Ramp0, Sim\_Generic/Ramp/Ramp1, Sim\_Generic/Ramp/Ramp2, Sim\_Generic/Ramp/Ramp3, Sim\_Generic/Ramp/Ramp5, and Sim\_Generic/Ramp/Ramp4. The "Target Name" column shows corresponding ramp tags (Ramp0-Ramp5). The "Data Type" column shows "Float8" for all items.
- Action/Trigger/Options:** A panel on the right showing "Execution Scheduling: Timer" (selected), "1 second(s)", "Data source: MySQL", "Table name: ramp\_historical", and "Automatically create table" checked.
- Database Query Browser:** A window titled "Database Query Browser" containing the SQL query "SELECT \* FROM ramp\_historical". Below the query, there is a "Limit SELECT to: 1000 rows" checkbox. The results are displayed in a "Resultset 1" table with columns: ramp\_historical\_..., Ramp0, Ramp1, Ramp2, Ramp3, Ramp5, Ramp4, and t\_stamp. The table shows 10 rows of data. The bottom of the browser shows "268 rows fetched in 0.013s" and buttons for Auto Refresh, Edit, Apply, and Discard.

## Database and OPC Synchronization

**Standard Groups** are the most flexible group. They are capable of not only storing OPC values in the database, but can also write database values to OPC addresses or synchronize data changes between both the database and PLC. With this group you can create true realtime value tables in the database, and allow anything that can talk to the database to push values to a PLC. This is often used to create **Recipe** stems where the recipe values are stored in the database, and a user can select a recipe to write all your settings directly to Tags. Changing recipes is as easy as changing a Tag value or selecting a name.

The screenshot shows a software interface for a 'Line 1 Recipe'. At the top, there are three buttons: 'Enabled' (green play icon), 'Disabled' (red stop icon), and 'Pause' (blue pause icon). Below this, the text 'Running' is displayed. The main area is titled 'Basic OPC/Group Items (4)' and contains a table with the following data:

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	

## Large Data Block Storage

Transfer large amounts of data very efficiently with the **Block Group**. This group allows you to send whole arrays of data to and from the database. It works just like the Standard group, but on a much larger scale.

The screenshot shows a software interface for 'T4 ACC'. At the top, there are three buttons: 'Enabled' (green play icon), 'Disabled' (red stop icon), and 'Paused' (blue pause icon). Below this, the text 'Execution Disabled' is displayed. The main area is titled 'Block Items (0)' and contains a table with the following data:

Item Name	Source ...	Latche...	Mode	Target Name	Data Type	Prop	Size
Item_T4_0	ns=1;s=[SLC]_Meta:T4/T4:0/T4:0.ACC	0	—	T4_0_ACC	String		2
	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 PLC data as inputs and outputs for your existing **Stored Procedures**. With the Stored Procedure Group, your IT department can have control over how data is entered and returned from the database.

SP All Params  
Errored

Enabled      Disabled      Pause

**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)**

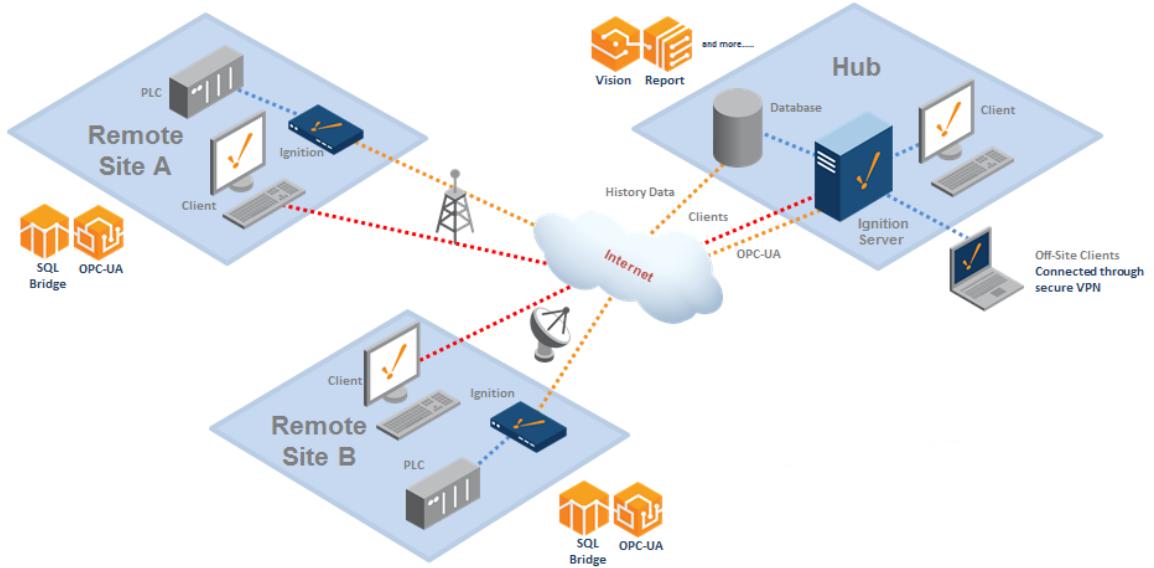
Item Name	Source Value	Latched Val...	Target Name	Data Type	Properties

## Centralizing Data Collection

In Distributed systems, PLCs can be spread out over great distances to remote sites. Collecting and centralizing data from each can be difficult and time consuming. To combat this problem, Transaction Groups are used as the cornerstone of our **Hub and Spoke** architecture. Historical Groups can be applied locally to each PLC for a minimal cost, and forward all data into a single, central, database.

### Remote Site with Central Hub

#### Remote Logging



In This Section ...

# Understanding Transaction Groups

Transaction Groups are the heart of the SQL Bridge module. They are units of execution that perform actions such as storing data historically, synchronizing database values to OPC, or loading recipe values. A variety of group types, items types, and options means that Transaction Groups can be configured to accomplish almost any task.

Transaction Groups are configured in the Ignition Designer. Each Transaction Group is associated with a table in a database Ignition is connected to, and is made up of one or more Items. The group will then execute at a specific interval of time, or on a user-defined schedule. Generally each execution will create a new row in the database table with a separate column for each Item in the group. However, it is possible for some types of Transaction Groups to take values from the database and write to specific Tags.

Additionally, the Transaction Group can be configured to conditionally synchronize values by using a trigger. The trigger is evaluated every execution, and if the trigger condition is met, then synchronization will occur. If the trigger condition has not been met, then the group will wait until the next execution to re-evaluate the trigger.

There are four [types of Transaction Groups](#): Standard, Block, Historical, and Stored Procedure. Each offers different functionality. For example, the Historical Group allows you to quickly configure a group that reads OPC data and push it to the database. While the additional flexibility of a Standard Group allows you take values from the database and write them to a PLC.

Transaction Groups enable you to perform tasks such as database to OPC synchronization, recipe management, and historical data logging.

## On this page

...

- [Transaction Group Workspace](#)
  - [Items](#)
  - [Enabling Group Execution](#)
  - [Editing Group Settings](#)
- [Action Settings](#)
- [Group Update Rate](#)
  - [Timer](#)
  - [Schedule](#)
  - [Execution Cycle](#)
- [Trigger and Handshake Settings](#)
- [Advanced Settings](#)
- [Creating a Transaction Group](#)

## 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:

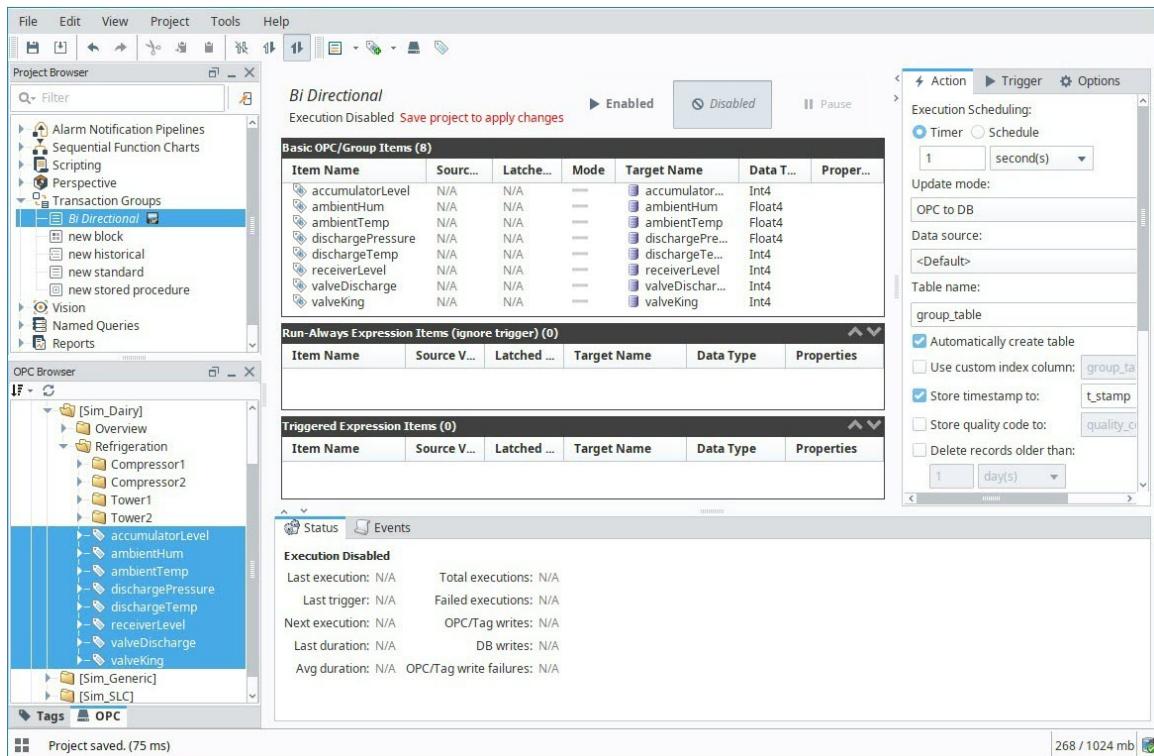
- **Title bar** - Shows the name of the currently selected group, as well as options to set it as Enabled or Disabled, and to Pause, if it's currently executing.
- **Item configuration** - Shows all of the items configured in the selected group. Many settings can be modified directly through the display, 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.



INDUCTIVI  
UNIVERSIT

## About Transaction Groups

[Watch the Video](#)



## 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.

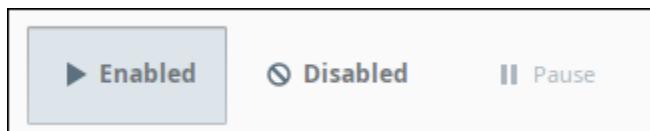
The Latched value will be the value that was written at execution. This can be the value that gets written to the database on execution in OPC to DB mode, or it can be the value that gets written to the Tag in DB to OPC mode. The value will only change on execution of the group.

### 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.





Transaction Groups exist in a project, but they execute in the global Gateway space. This means that once your groups are enabled, they will run even without a client open.

## Editing Group Settings

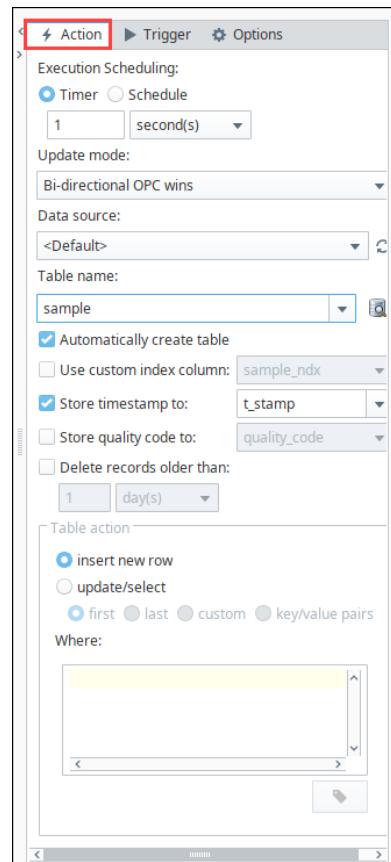
Group settings may be modified at any time, regardless of whether or not the group is executing. Modifications will be applied when the project is saved, and the group will be started or stopped as required. Some changes such as modifying items may cause features like live values to appear to be incorrect. It is therefore important to note 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.

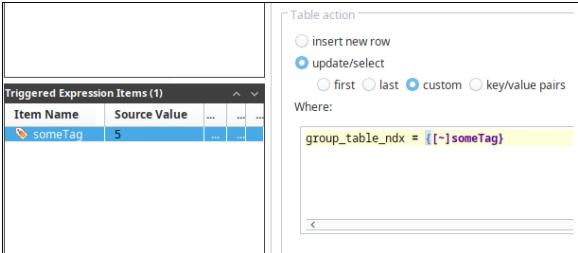
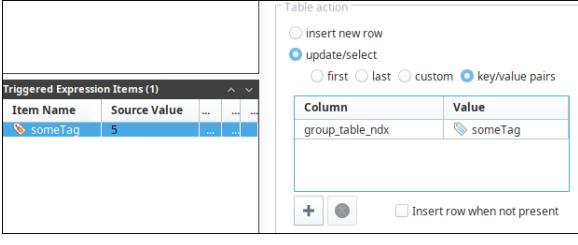
## Action Settings

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:

Setting	Description
Execution scheduling	How often the group is evaluated. For a number of reasons, the group may not execute during the evaluation. The most common reason is the trigger, but see <a href="#">Execution Cycle</a> below for more possible reasons why evaluation will exit. <ul style="list-style-type: none"><li><b>Timer</b> - specifies the OPC Tag subscription rate for the OPC Tags. It can run at millisecond, second, minute, hour, or day rates.</li><li><b>Schedule</b> - is a specified start time on the update <b>Rate</b>. Set 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 execute as in timer mode during that period.</li></ul>
Update mode	For groups that support it, sets the default for how items are compared to their targets. Options are: <ul style="list-style-type: none"><li>OPC to DB - Only read from the <a href="#">OPC server</a> and write to the <a href="#">database</a>.</li><li>DB to OPC - Only read from the <a href="#">database</a> and write to the <a href="#">OPC Server</a>.</li><li>Bi-directional OPC wins - Read and Write to both the <a href="#">database</a> and <a href="#">OPC Server</a>. On group start, write <a href="#">OPC</a> values to the <a href="#">database</a>.</li><li>Bi-directional DB wins - Read and Write to both the <a href="#">database</a> and <a href="#">OPC Server</a>. On group start, write <a href="#">database</a> values to <a href="#">OPC</a> items.</li></ul>
Data source	The database connection name the group should use. Can be <b>Default</b> , 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, depending on the <b>Update mode</b> and individual item <b>Mode</b> settings). The tables listed in this dropdown are determined by the <b>Data source</b> 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 <b>Automatically create table</b> 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 <b>Table name</b> 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 <b>Automatically create table</b> 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 <b>Automatically create table</b> setting.
Store quality code	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.
Delete records older than	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)
Table Action	<p>Defines which row will be targeted by the group.</p> <ul style="list-style-type: none"> <li>• <b>insert new row</b></li> <li>• <b>update/select</b> - allows you to target specific rows in the database table. Options are: <ul style="list-style-type: none"> <li>◦ <b>first row</b> - the group always executes against the first row.</li> <li>◦ <b>last row</b> - the group always executes against the last row</li> <li>◦ <b>custom</b> - allows you to write a custom where clause to determine which row should be targeted. Uses the <b>Where</b> text area. The custom clause can use references to values of items in the group.</li> </ul> </li> </ul>  <p>◦ <b>key/value pairs</b> - 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. In the image below, a value of 5 will be used in conjunction with the "group_table_ndx" column in the database table. Additional conditions can be added or removed with the Add or Delete buttons, below the table.</p> <p>Meaning, when the group executes, it will target the row where group_table_ndx has a value of 5.</p> 

## 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.

### Timer



**INDUCTIVI**  
**UNIVERSIT**

**Group Update Rate**

[Watch the Video](#)

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 execute 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. Ie. "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 execute 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 core evaluation may differ, but the general cycle is the same.

1. Timer executes, group enters execution
2. Is the group paused? Break execution.
3. Is the Gateway part of a redundant pair? If so, is it active? If not active, break execution.  
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, break execution.
6. Evaluate "triggered" expression items.
7. If applicable, read values from the database.
8. Execute a comparison between items and their targets.
9. Execute any writes to other Tags or the database that results from execution.
10. Report alerts.
11. Acknowledge the trigger, if applicable.
12. Write handshake value, if applicable.

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



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 and Handshake Settings

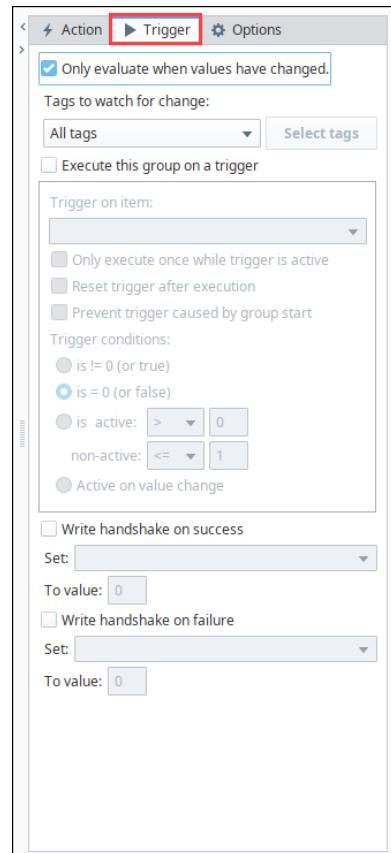
The trigger settings determine when a group will actually execute. They are examined each time the group evaluates (according to the update rate of the group). If they pass, the group will run and perform its action against the database.

The trigger settings are the same for all group types and are found on the second tab (labeled **Trigger**), on the right side of the Transaction Group workspace.

The outcome of an execution is handled in the handshake section of the trigger section of the transaction group. When a group executes, it either completes successfully or an error prevents its execution.

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

Setting	Description
Only evaluate when values have changed	<p>The group will execute every time the value or values change. If the values have not changed, it will exit the evaluation. Note, it is possible to monitor all <b>Run-Always</b> items in the group, or only specific ones.</p> <ul style="list-style-type: none"> <li><b>Tags to watch for change</b> - Executes on 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.</li> </ul>
Execute this group on a trigger	<p>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.</p> <ul style="list-style-type: none"> <li><b>Trigger on item</b> - select the item time you want to use as the trigger.</li> </ul>
Only execute once while trigger is active	<p>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 time the trigger conditions evaluate to true.</p>
Reset trigger after execution	<p>If using the "&gt;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.</p>
Prevent trigger caused by group start	<p>If selected, the group will not execute if the trigger is active on the first evaluation of the group. In the course of designing a group, it is common to stop and start it many times, and sometimes it is not desirable to have the group execute as a result of this. Selecting this option will prevent these executions, as well as executions caused by system restarts.</p>
Trigger conditions	<p>Set any of the following trigger conditions:</p> <ul style="list-style-type: none"> <li>is !=0 (or true)</li> <li>is =0 (or false)</li> <li>is active or non-active, which causes the group to execute if the trigger value matches the <b>is active</b> condition.</li> <li>Active on value change, which will cause the group to execute if the trigger changes value at all.</li> </ul>
Write handshake on success	<p>Set the item and the value you want to see when the group executes successfully.</p>
Write handshake on failure	<p>Set the item and the value you want to see when an error prevents the group execution.</p>

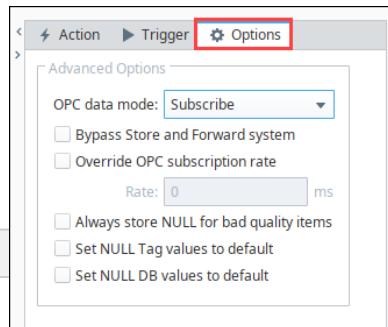


To learn more about configuring Transaction Groups with the different trigger options, refer to the [Trigger Options](#) page.

# 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.

Setting	Description						
OPC Data Mode	<p>Modifies how the group receives data from OPC.</p> <table border="1"> <thead> <tr> <th>Option</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Subscribe</td><td> <p>- 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 performance, as it reduces unnecessary data flow and allows the OPC server to optimize reads.</p> <p><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.</p> </td></tr> <tr> <td>Read</td><td> <p>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.</p> </td></tr> </tbody> </table>	Option	Description	Subscribe	<p>- 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 performance, as it reduces unnecessary data flow and allows the OPC server to optimize reads.</p> <p><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.</p>	Read	<p>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.</p>
Option	Description						
Subscribe	<p>- 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 performance, as it reduces unnecessary data flow and allows the OPC server to optimize reads.</p> <p><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.</p>						
Read	<p>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.</p>						
Bypass Store and Forward System	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.						
Override OPC subscription rate	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.						
Always store NULL for bad quality items	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.						
Set NULL Tag values to default	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.						



Set  
NULL  
DB  
values  
to  
default

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.

The following feature is new in Ignition version **8.0.11**  
[Click here](#) to check out the other new features

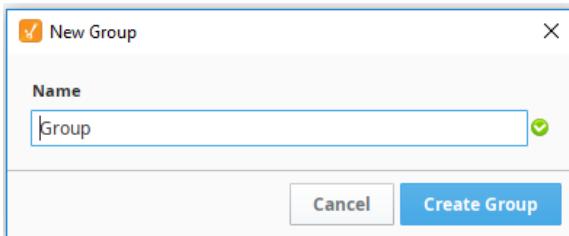
As of 8.0.11, enabling the Set DB Values to Default setting on Block Groups will clear the latched value, setting the item to a default if the corresponding database value is Null.

Type	Default Value
Byte	0
Short	0
Integer	0
Long	0
Float	0.0
Double	0.0
Boolean	FALSE
String	" (Empty String)
Date/Time	Current Date/Time
Dataset	[0x0] (Empty Dataset)
Array	[] (Empty Array)

## 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.'



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



INDUCTIVI  
UNIVERSIT

### Basic Historical Group

[Watch the Video](#)

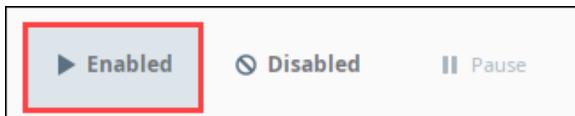


INDUCTIVI  
UNIVERSIT

### Realtime Group

[Watch the Video](#)

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".
- Note that right now your group only exists in the Designer.

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.

Database Query Browser

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

Limit SELECT to: 1,000 rows

**Resultset 1**

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

Auto Refresh | Edit | Apply | Discard

MySQL

Schema History

- group\_table3
- group\_table\_new
- history
- history\_sine\_tags
- hitachi\_errors
- hitachi\_messages

Related Topics ...

- Database Connections
- Transaction Group Examples
- Database Query Browser
- Trigger Options

In This Section ...

# 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.

## Historical Group

The historical group makes it easy to quickly log data historically to a SQL database.

### General Description

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.

## Standard Group

The Standard Group is called such because it's a flexible, general use group that can be adapted to a variety of situations. The data model is row based, with items mapping to columns and the data corresponding to a specific row of a table.

### General Description

The Standard Group contains items, which may be mapped to the database, 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. Items can be mapped in a one-way fashion, or bi-directionally, in which the value of the database and the item will be synchronized.

The group may also insert new rows instead of updating a specific row. In this manner, data can be inserted for historical purposes based on a timer, with an optional trigger.

### 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.

## On this page

...

- Historical Group
  - General Description
  - Group Settings
  - Typical Uses
- Standard Group
  - General Description
  - Group Settings
- Table action
  - Typical Uses
- Block Group
  - General Description
  - Typical Uses
  - Table Format
  - Row ID and Block ID
  - Group Settings
  - Table action
- Stored Procedure Group
  - Group Settings
  - Typical Uses
  - Known Issues
  - Parameters in the Stored Procedure Group



### Types of Groups

[Watch the Video](#)

**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, 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 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.

## 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.

## General Description



Block Group

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 you to 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 (int, 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 like so:

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

Note that 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:

[Watch the Video](#)

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 look 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 is 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 index**. 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.

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).



### Stored Procedure Group

[Watch the Video](#)

## 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	<p>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.</p> <p>Expression items have two sub types:</p> <ul style="list-style-type: none"><li>• <b>Run-Always</b> expression items are evaluated every time the group executes. Meaning, they'll run their associated expression or query every time the group executes.</li><li>• <b>Triggered</b> expression items only evaluate when the group trigger is active.</li></ul>
Tag Reference Item	<p>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 <a href="#">Tag References vs. OPC Items</a> section on this page.</p> <p>Tag Reference Items can reference the value on any Tag in a Tag provider, such as query Tags and memory Tags.</p>

## On this page

...

- Tag References and OPC Items
- Expression Items
  - Scope
  - Execution Order
  - Expression Type
- Run Always vs. Triggered Items
  - Changing the Evaluation State
- SQL Queries and Expressions
- Creating a New Item
- Item Type Property Table
  - OPC Item Options
  - Tag Reference Item Options
  - Expression Item Options



INDUCTIVE  
UNIVERSITY

## 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.



INDUCTIVE  
UNIVERSITY

## Tag References vs. OPC Items

[Watch the Video](#)

Refer to the Properties Table at the bottom of this page to see the properties for both [Tag](#) and [OPC Items](#).

## 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.

### Scope

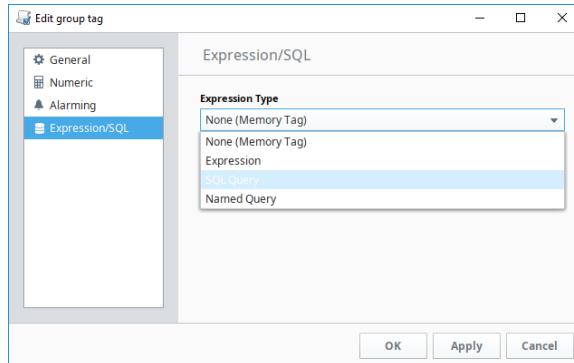
It is important to understand that an Expression Item only exists within its group, and can not be referenced by items in other Transaction Groups, Tags, and any components on a window. The only exception is to use an Expression Item to store/retrieve a value with `storeVariable()` and `getVariable()` functions. These functions store values in a space that is accessible by all Transaction Groups.

### 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	Definition
None	Behaves similar to a Memory Tag in that the value does not automatically change.
Expression	Uses Ignition's <a href="#">Expression Language</a> 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 properties](#).



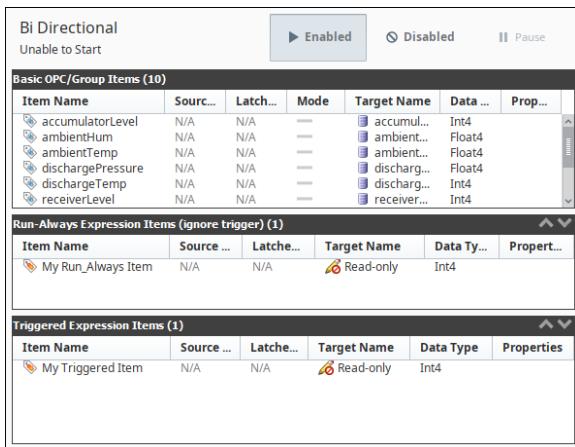
### Expression Items

[Watch the Video](#)

# 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 because it always runs.



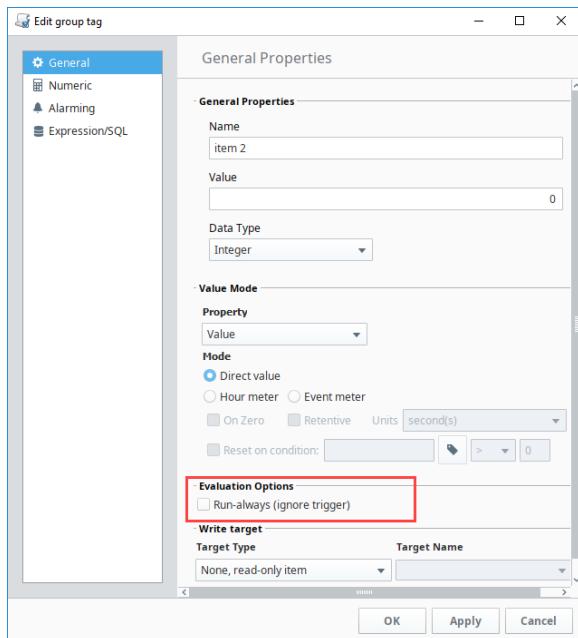
INDUCTIVE  
UNIVERSITY

## Run Always vs. Triggered Items

[Watch the Video](#)

## 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

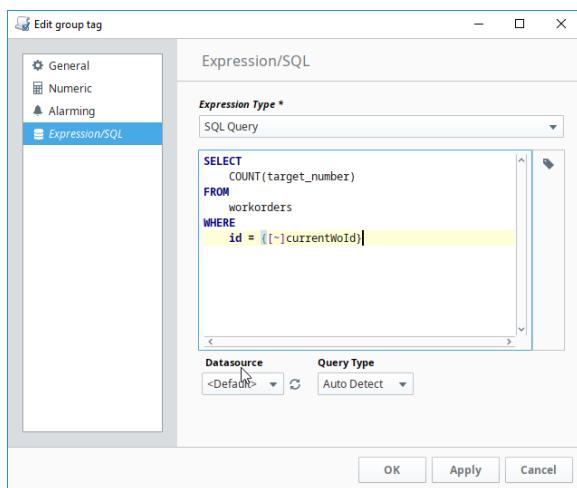


## SQL Queries and Expressions

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.



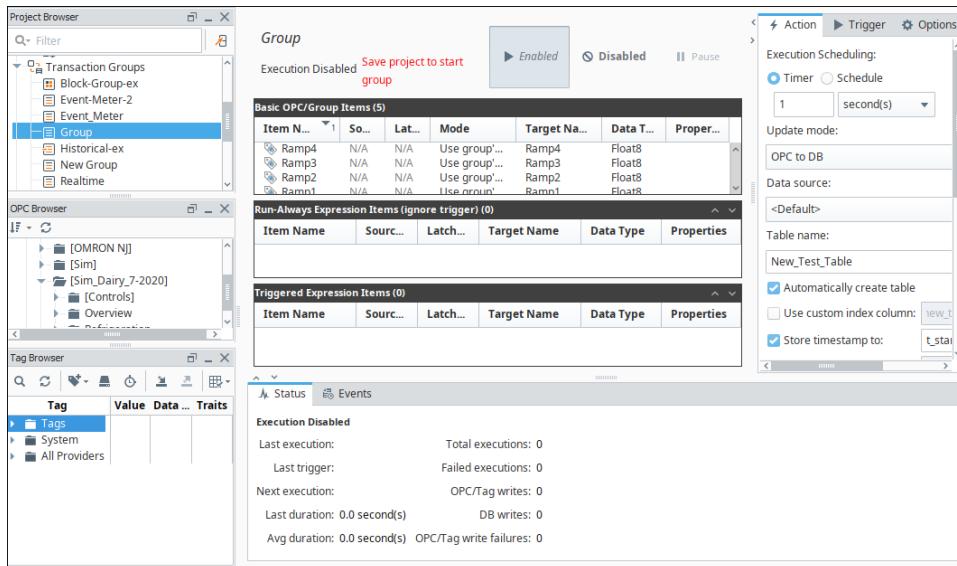
## SQL Query Expression Items

[Watch the Video](#)

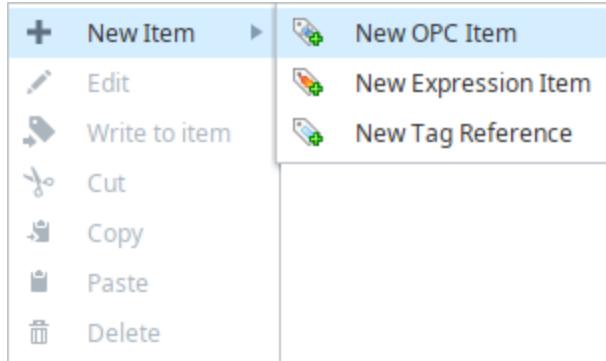
## Creating a New Item

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:



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 [property table](#) on this page for more information on the various item types and their properties.



4. Once you configured the item, click **OK**. Different items have different properties. A description of [item properties](#) for each type can be found on this page.

## Item Type Property Table

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

### OPC Item Options

Property	Description
<b>General</b>	
Name	The name of the OPC item in the group. There cannot be duplicate names within a group.
Data Type	The datatype used to read values from the PLC.
<b>OPC Properties</b>	
OPC Server	The Selected OPC Server. This is a drop-down 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.
Source Data Type	Data type for the OPC item.
<b>Value Mode</b>	
Property	Which property of the OPC item you want to use. <ul style="list-style-type: none"> <li>• Value - Item value</li> <li>• Quality - Quality code from OPC Server (192 = GOOD_DATA)</li> <li>• Timestamp - The last time the item value changed</li> <li>• Name - The SQLBridge Item Name property of this Item</li> </ul>

Mode	<p>Options for displaying values based on the Item value.</p> <ul style="list-style-type: none"> <li>• Direct Value - Item value</li> <li>• 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 datatype should be set to integer or float when using an Hour Meter regardless of the OPC Item type.           <ul style="list-style-type: none"> <li>◦ On Zero - Use a zero value to accumulate time instead of a non-zero value</li> <li>◦ Retentive - Retain the Hour Meter value when it is not accumulating.</li> <li>◦ Units - The time units to display.</li> </ul> </li> <li>• Event Meter - Record the number or times the Item value is non-zero. The datatype should be set to integer when using an Event Meter regardless of the OPC Item type.           <ul style="list-style-type: none"> <li>◦ On Zero - Use a zero value to accumulate events instead of a non-zero value</li> </ul> </li> </ul>
------	--

## Write Target

Mode	<p>Changes the items directional read/write option.</p> <ul style="list-style-type: none"> <li>• Use group's mode - Inherit the Update Mode from the Item's Group.</li> <li>• OPC to DB - Only read from the OPC server and write to the database.</li> <li>• DB to OPC - Only read from the database and write to the OPC Server.</li> <li>• Bi-directional OPC wins - Read and Write to both the database and OPC Server. On group start, write OPC Server values to the database.</li> <li>• Bi-directional DB wins - Read and Write to both the database and OPC Server. On group start, write database values to the OPC Server.</li> </ul>
Target Type	<p>This is the selection for what the Item will write to when the group executes.</p> <ul style="list-style-type: none"> <li>• None, read-only item - Do not write this value to the database.</li> <li>• 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.</li> </ul>
Target Name	<p>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.</p>
Alarming	<p>The Alarming settings for the OPC items. See <a href="#">Alarming Properties</a> for a full explanation.</p>

## 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 datatype to write to into the database if this item is not read-only.
Value Mode	
Property	<p>Which property of the Tag you want to use.</p> <ul style="list-style-type: none"> <li>• Value - Item value</li> <li>• Quality - Quality code of the Tag (192 = GOOD_DATA)</li> <li>• Timestamp - The last time the item value changed</li> <li>• Name - The SQLBridge Item Name property of this Item.</li> </ul>

Mode	<p>Options for displaying values based on the Item value.</p> <ul style="list-style-type: none"> <li>• Direct Value - Item value</li> <li>• 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 datatype should be set to integer or float when using an Hour Meter regardless of the OPC Item type.</li> <li>• On Zero - Use a zero value to accumulate time instead of a non-zero value</li> <li>• Retentive - Retain the Hour Meter value when it is not accumulating.</li> <li>• Units - The time units to display.</li> <li>• Event Meter - Record the number or times the Item value is non-zero. The datatype should be set to integer when using an Event Meter regardless of the OPC Item type.</li> <li>• On Zero - Use a zero value to accumulate events instead of a non-zero value</li> </ul>
------	--

## Write Target

Mode	<p>Changes the items directional read/write option. This is only editable when the target Type is set to Database field.</p> <ul style="list-style-type: none"> <li>• Use group's mode - Inherit the Update Mode from the Item's Group.</li> <li>• OPC to DB - Only read from the OPC server and write to the database.</li> <li>• DB to OPC - Only read from the database and write to the OPC Server.</li> <li>• Bi-directional OPC wins - Read and Write to both the database and OPC Server. On group start, write OPC Server values to the database.</li> <li>• Bi-directional DB wins - Read and Write to both the database and OPC Server. On group start, write database values to the database.</li> </ul>
Target Type	<p>This is the selection for what the Item will write to when the group executes.</p> <ul style="list-style-type: none"> <li>• None, read-only item - Do not write this value to the database.</li> <li>• Database field - Write the Item value to the specified column in the database table.</li> </ul>
Target Name	<p>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.</p>

## 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.
Datatype	The datatype values are stored as.
Value Mode	
Property	<p>Which property of the OPC item you want to use.</p> <ul style="list-style-type: none"> <li>• Value - Item value</li> <li>• Quality - Quality code of the expression/SQL Query (192 = GOOD_DATA)</li> <li>• Timestamp - The last time the item value changed.</li> <li>• Name - The SQLBridge Item Name property of this Item.</li> </ul>

Mode	<p>Options for displaying values based on the Item value.</p> <ul style="list-style-type: none"> <li>• Direct Value - Item value</li> <li>• 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 datatype should be set to integer or float when using an Hour Meter regardless of the OPC Item type.</li> <li>On Zero - Use a zero value to accumulate time instead of a non-zero value</li> <li>Retentive - Retain the Hour Meter value when it is not accumulating.</li> <li>Units - The time units to display.</li> <li>• Event Meter - Record the number or times the Item value is non-zero. The datatype should be set to integer when using an Event Meter regardless of the OPC Item type.</li> <li>On Zero - Use a zero value to accumulate events instead of a non-zero value</li> </ul>
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	<p>Target Type - This is the selection for what the Item will write to when the group executes.</p> <ul style="list-style-type: none"> <li>• None, read-only item - Do not write this value to the database.</li> <li>• Database field - Write the Item value to the specified column in the database table.</li> <li>• Other Tag - Write the Expression Item's value back to an OPC item or Tag Reference.</li> </ul>
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 <a href="#">Tag Scaling Properties</a> .
Alarming	These are the Alarming settings for the OPC items. See <a href="#">Alarming Properties</a> for a full explanation.
Expression	These are the Expression/SQL Query options for Expression Items. See <a href="#">Expression/SQL Properties</a> 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.

## 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.

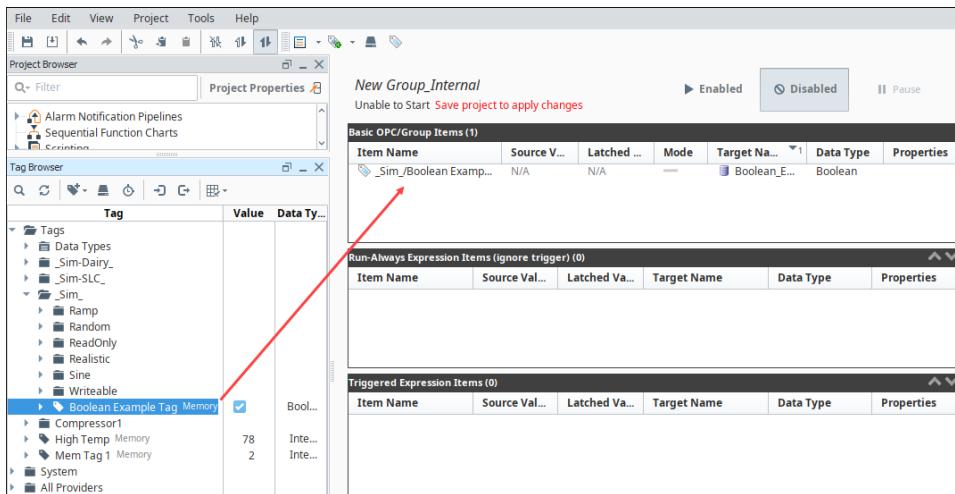


## Hour and Event Meters

[Watch the Video](#)

## Count the Duration of 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**.
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

Enabled      Disabled      Pause

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 Exam...	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

Context menu for the second item: New Item, Edit (highlighted), Write to item, Cut, Copy, Paste, Delete.

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 (radio button)

Hour meter (radio button, highlighted)

Event meter

On Zero:  Retentive:  Units: second(s)

Reset on condition:  (dropdown menu)

- Write target

Mode: Use group's mode

Target Type: Other tag (highlighted)

Target Name: \_Sim/\_Ramp/Ramp0 (highlighted)

OK      Apply      Cancel

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:

- 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.

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 Example...	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

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

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 Example...	N/A	N/A	---	Read Only	Boolean	

Run-Always Expression Items (ignore trigger) (0)					
Item Name	Source Val...	Latched Va...	Target Name	Data Type	Properties

- Click **Enabled** at the top of the page, and do a **File > Save** to start the group.
- 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

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

► Enabled  Disabled  Pause

**Basic OPC/Group Items (2)**

Item Name	Source...	Latched ...	M...	Target ...	↑ Data Type	Properties
_Sim_New_Programmable_/_Boolean1	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

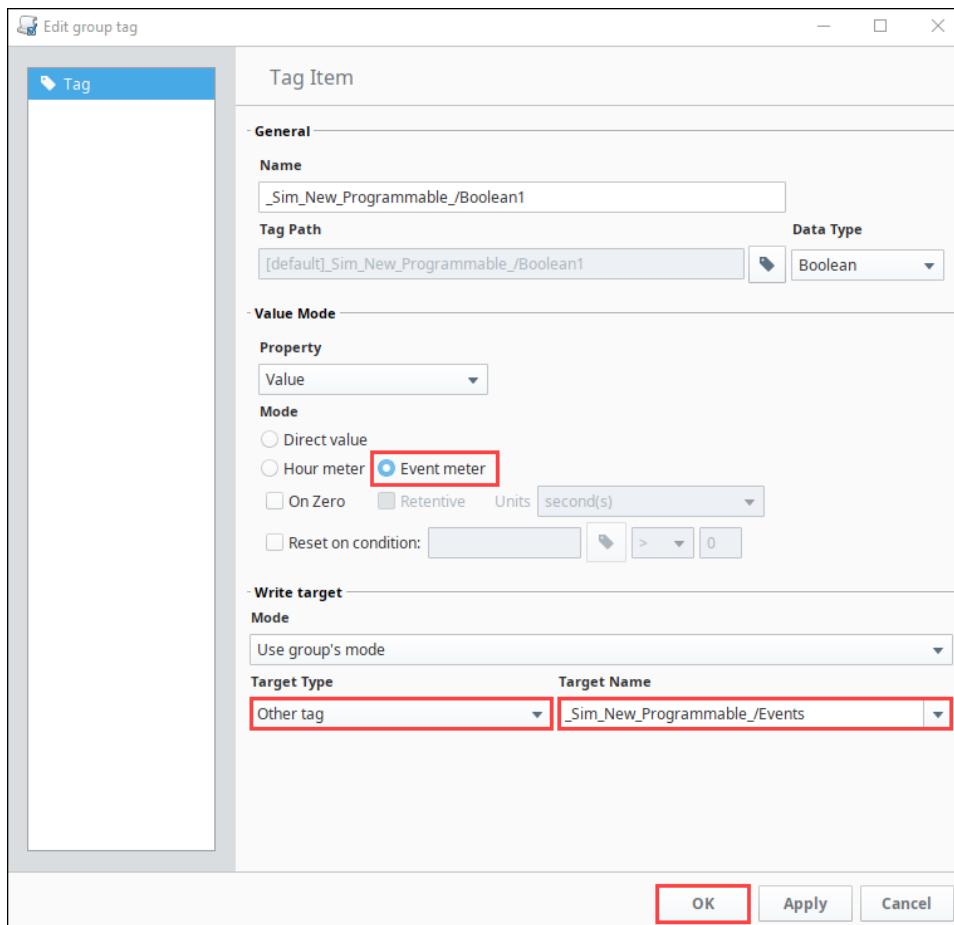
3. Right-click on the boolean Tag in the Transaction Group and select **Edit** to edit it.
4. In the **Edit group tag** window, set the following:

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**.



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.

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

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.

Item Name	Source Value	Latched Va...	Target Name	Data Type	Properties
Boolean1	1			Boolean	
Events	1			Integer	
Machine On				Boolean	

## Reset an Hour or Event Meter Based on a Condition

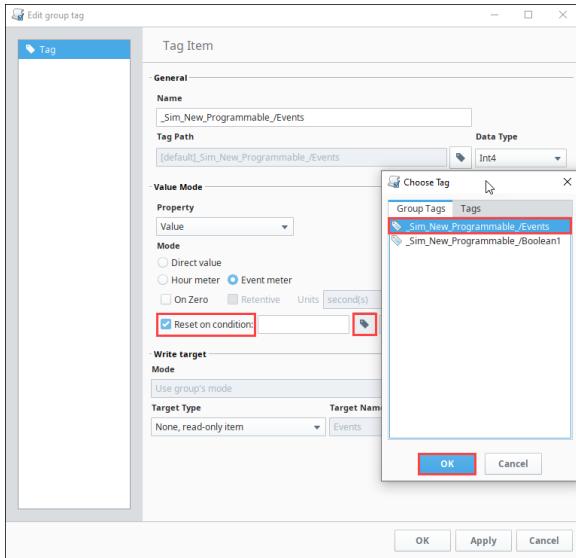
You can set the hour or event meter based on a condition.

1. In the Basic OPC/Groups 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**.

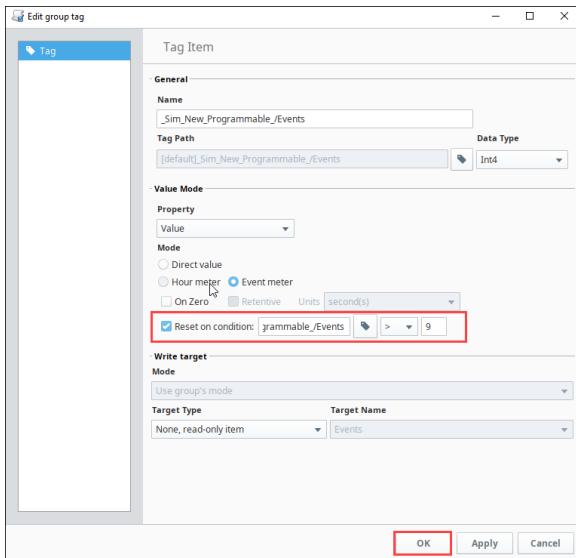


### Resetting Hour and Event Meters

[Watch the Video](#)



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



6. Click **OK**. The target Tag will now reset in response to the condition (after 9 occurrences in our example).

## Next...

- o [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](#).

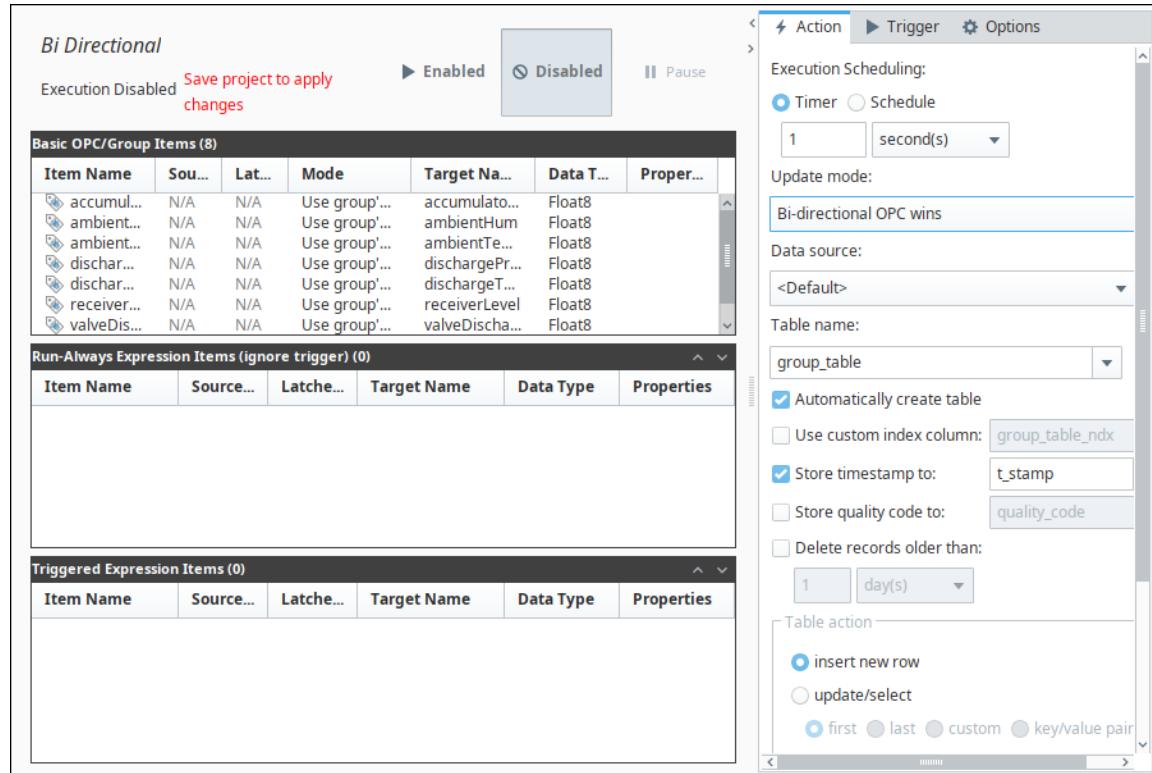
### On this page

...

- Transaction Groups
- Standard Group
- Historical Group
- Block Group
- OPC to OPC Interaction

## 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.



## 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.

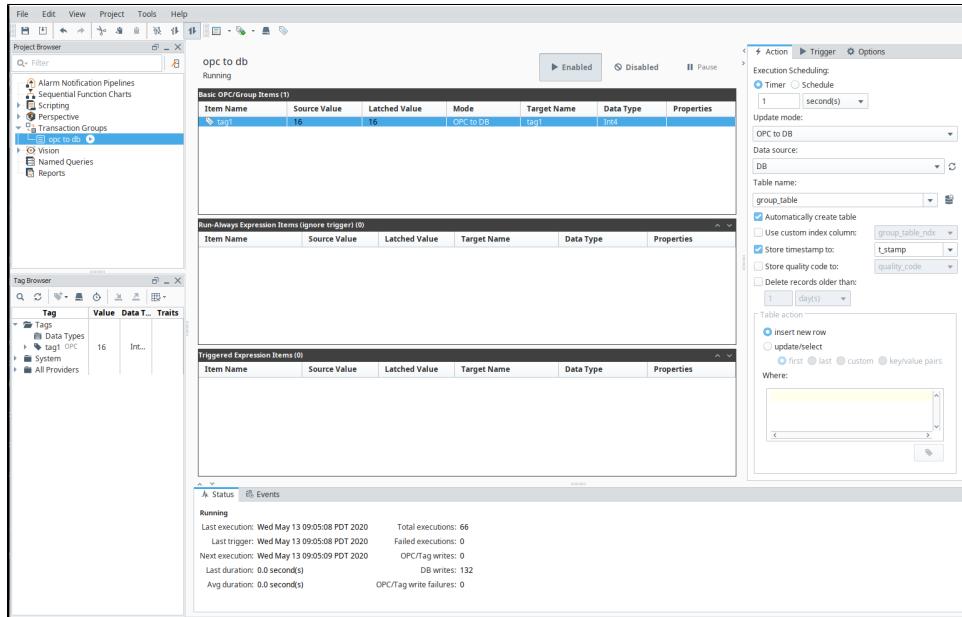
## Block Group

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.

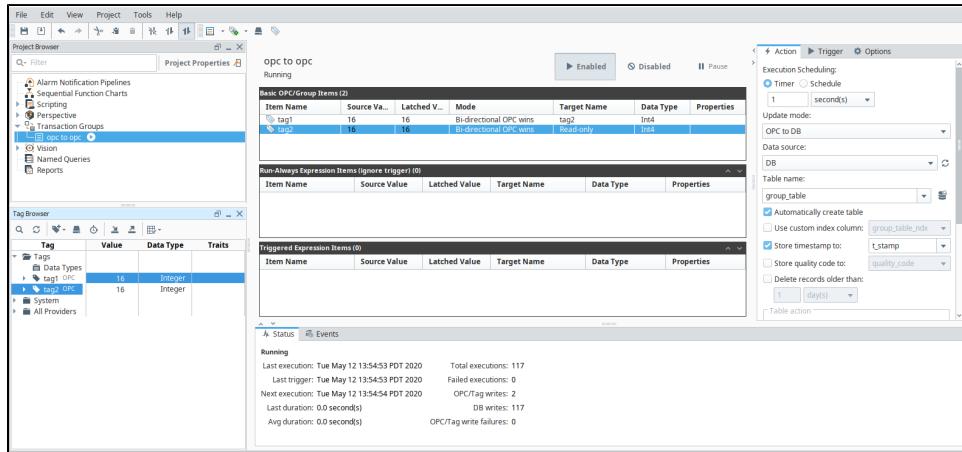
## OPC to OPC Interaction

It is 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, 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 PLC's.
2. Set the Mode on 'tag1' to **'Bi-directional OPC Wins'** and set its Target Name to be 'tag2.'
3. 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. Your configuration should match what is shown below:



What this will do is make sure that every 1 second, the value from 'tag1' will be written to 'tag2' as below:



## 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

...

- Create a Block Group
  - DB to OPC Mode with Custom Where Clause
- Next...

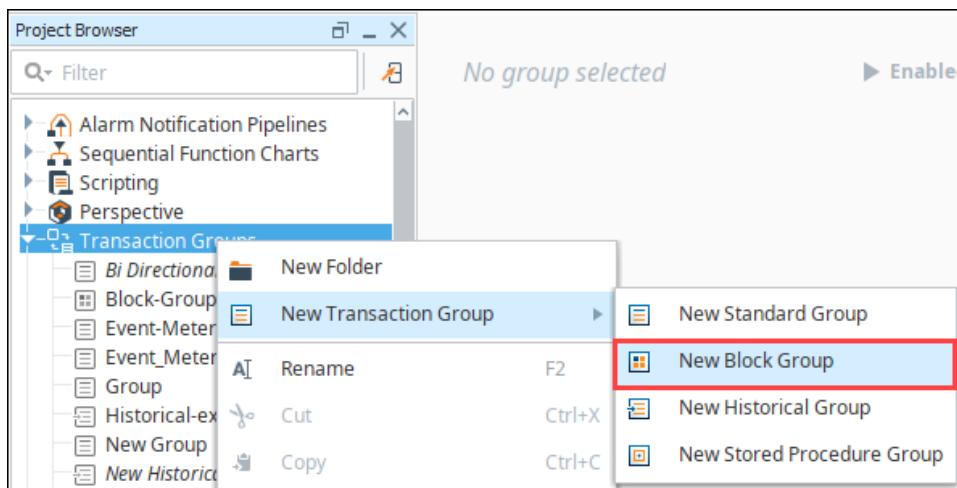


## Block Group

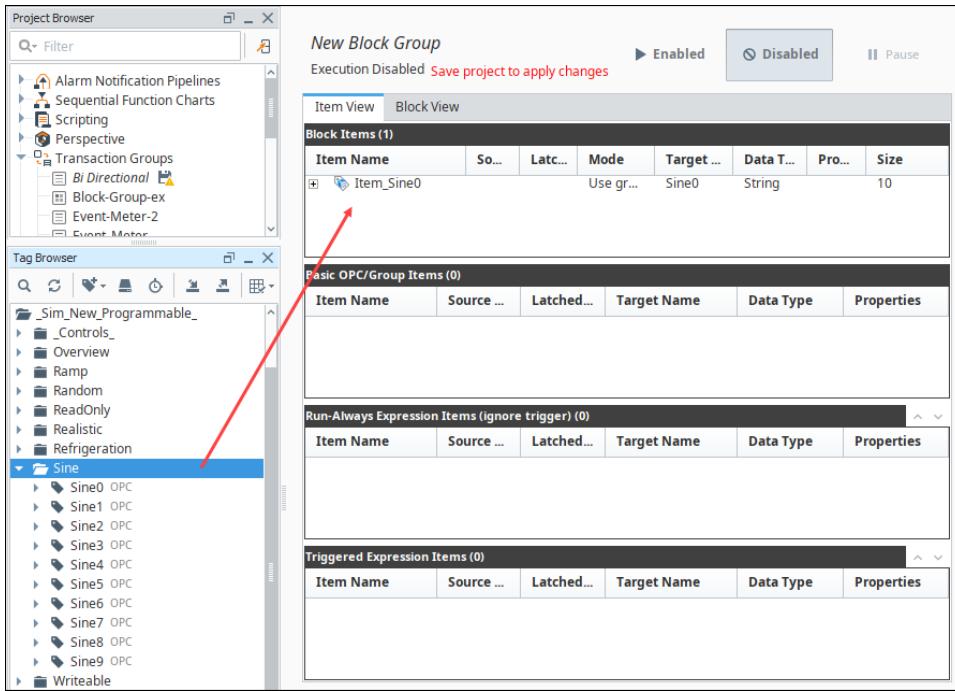
[Watch the Video](#)

## Create a Block Group

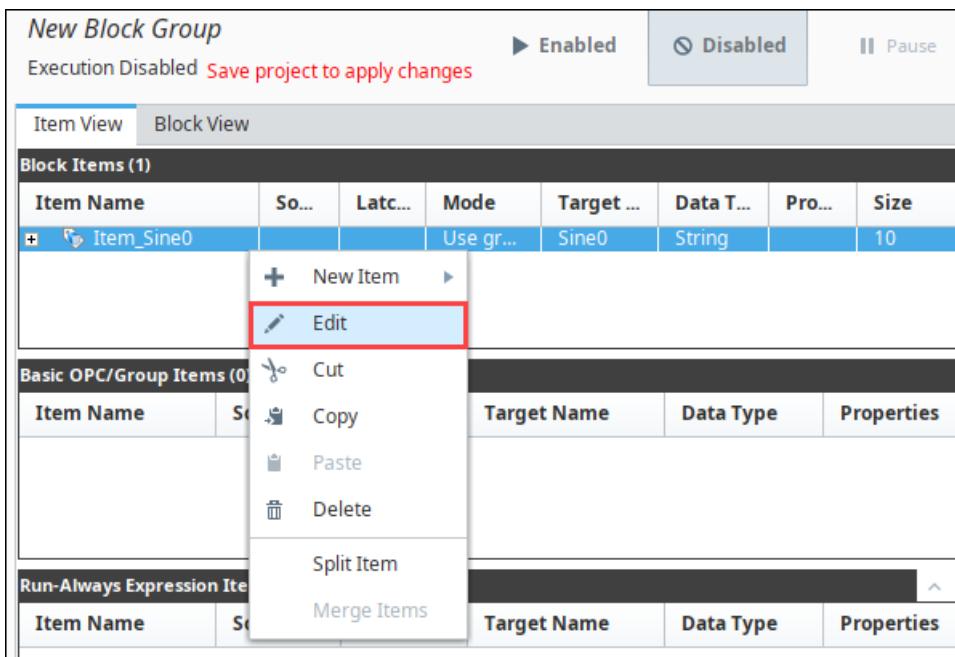
1. In the Project Browser, right-click on **Transaction Groups** and select **New Transaction Group > New Block Group**.



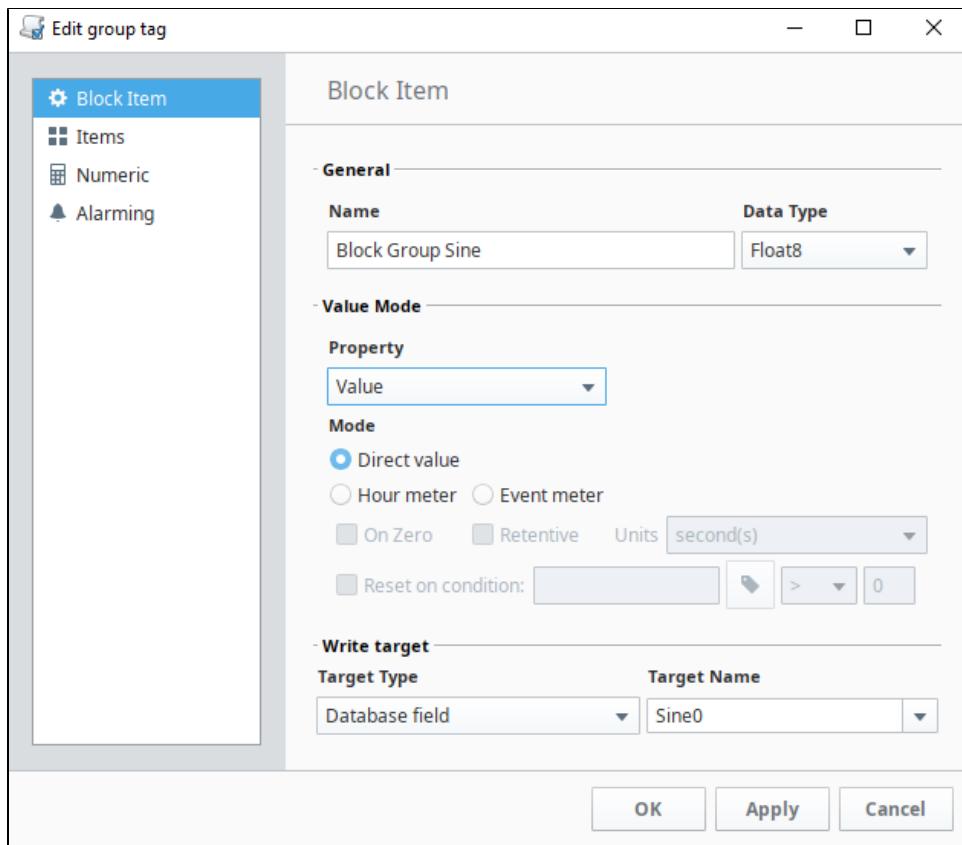
2. Give the group a name and click **Create Group**.
3. Drag a Tag folder into the **Block Items** section of the new Transaction Group.



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



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



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

**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

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:

Where condition editor

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 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



The screenshot shows a block configuration interface. At the top, there is a status bar with 'Block' and two radio buttons: 'Enabled' (selected) and 'Disabled'. Below this is a 'Execution Disabled' message. The main area has tabs for 'Item View' (selected) and 'Block View'. Under 'Block Items (1)', there is a table with the following data:

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	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	7,000,000	Integer		
itemValue1 Memory	50,000	Integer		
itemValue2 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	7000000 50000 600000	7000000 50000 600000		itemValue	Int4		3

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.

## 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 that 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.

<b>id</b>	<b>recipe_name</b>	<b>setpoint1</b>	<b>setpoint2</b>
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



## Recipe Group

[Watch the Video](#)

## 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 Browser

Tag	Data Type
Tags	
Data Types	
_Generic_Pri	
_Sim_New_P	
Sim_Dairy_7	
Tanks_OP	
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](#) Enabled Disabled Pause

Item Name	Source...	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	

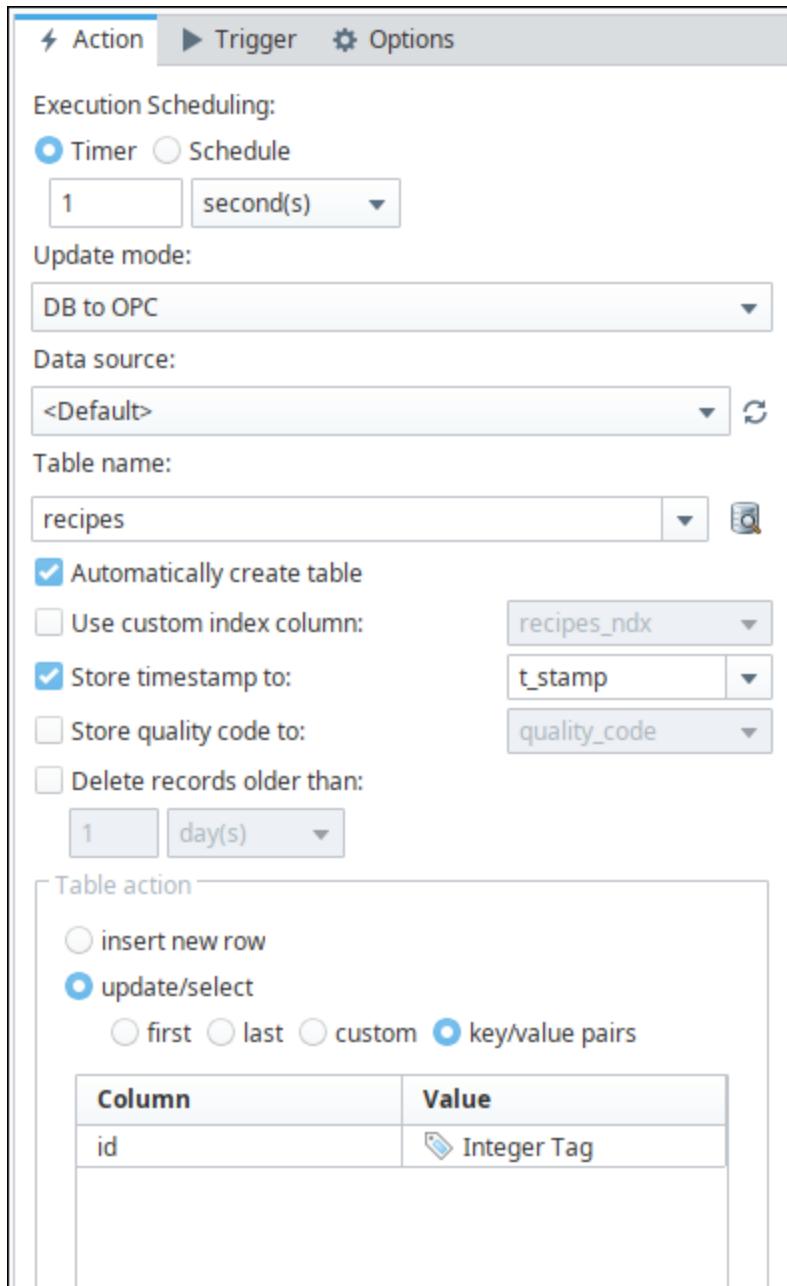
Run-Always Expression Items (ignore trigger) (0)

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

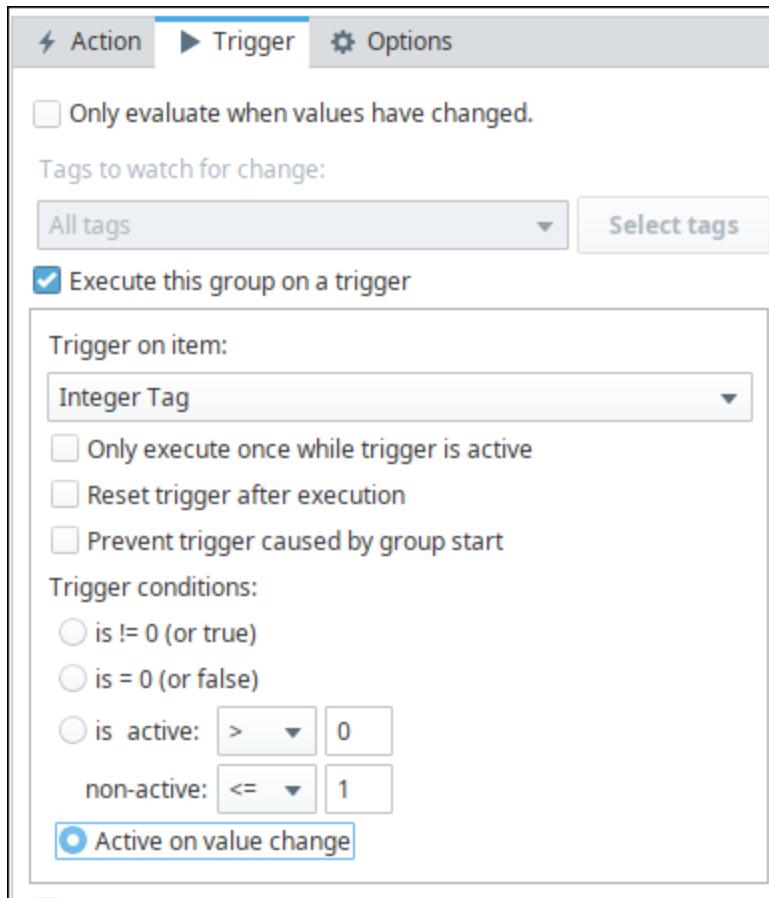
3. Set the Table Name to 'recipes', the table that we created earlier.
4. We then need to ensure that our Tags will be receiving the proper values from the database. Set the Target Names for each of the Tags: the string to 'recipe\_name', the floats to 'setpoint1' and 'setpoint2', and the integer to '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.



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**.



- 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.
- 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.

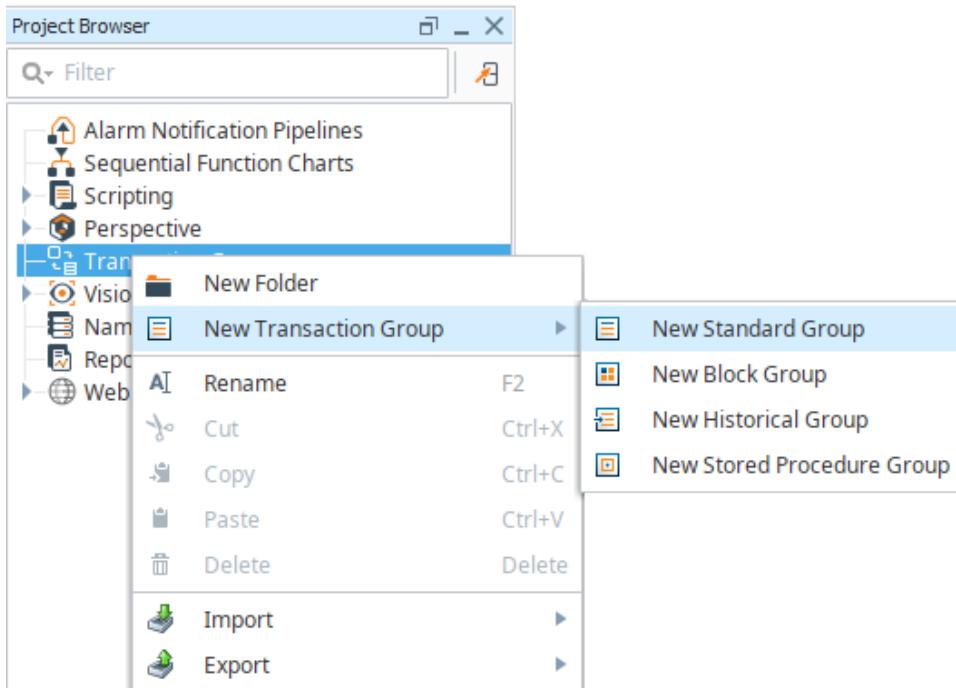


## Update or Insert Group

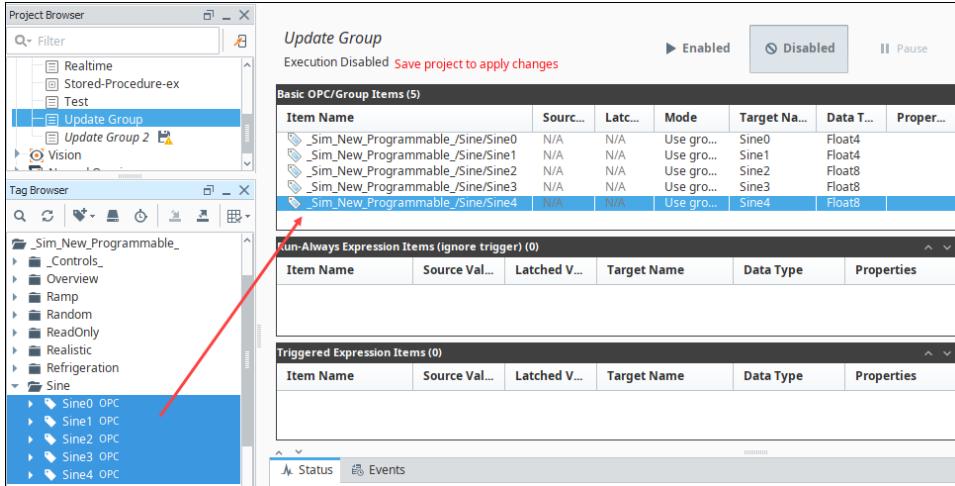
[Watch the Video](#)

## Update or Insert a New Row into the Database

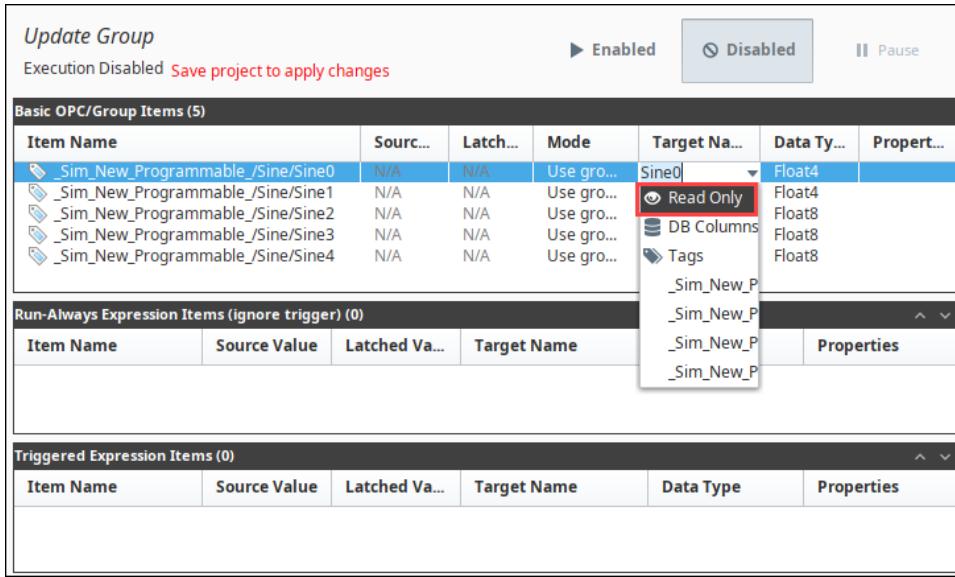
1. In the Project Browser, right-click on Transaction Groups and select **New Standard Group**.



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



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



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.

**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)

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.

## 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.

### On this page

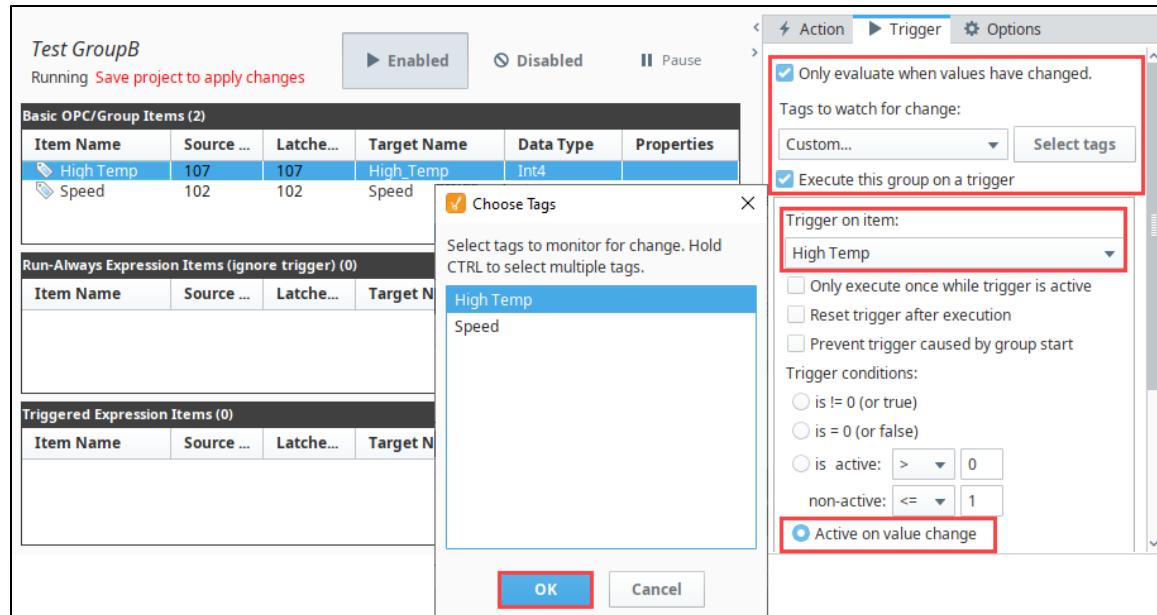
...

- Execute on Value Change
- Execute while Condition Is True
- Execute on a Rising Edge
- Reset Trigger
- Handshakes
- Next...



### Trigger – On Value Change

[Watch the Video](#)



## 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**.
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.

## 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.

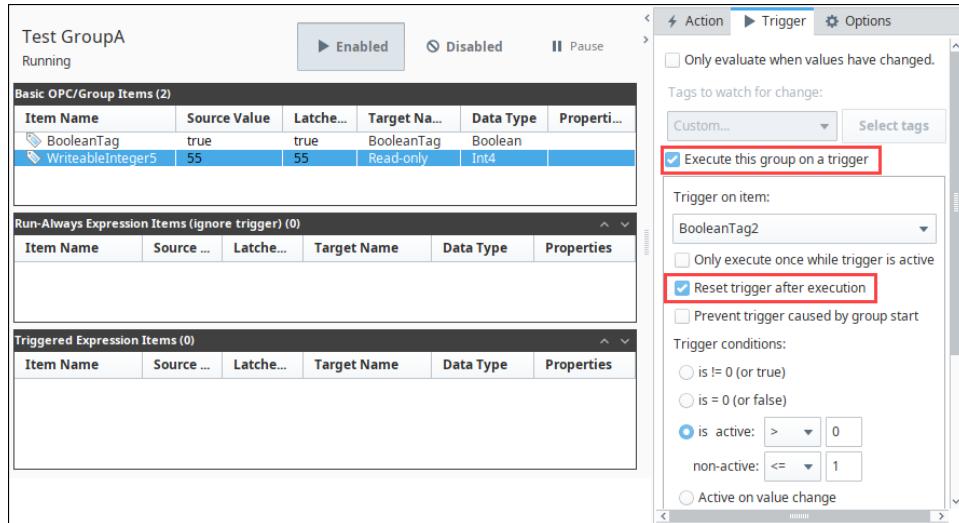
1. Create a standard Transaction Group with any number of Tags as long as one of them is a boolean Tag that will serve as the trigger for the group.
2. Set the **Write Target** for the boolean Tag to **Read-only** by selecting read-only from its drop down in the Target Name column.
3. Go to the **Trigger** 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.

## 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 write target for this Tag should be read only.
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.



## 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

► 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

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.

## On this page

...

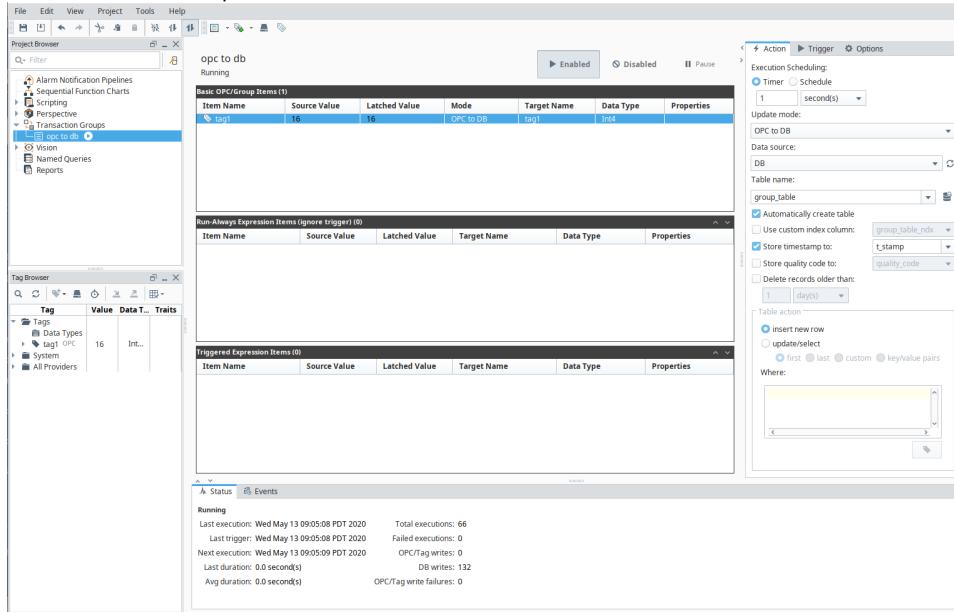
- **OPC to DB**
- **DB to OPC**
- **Bi-directional OPC Wins**
- **Bi-directional DB Wins**

## 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.

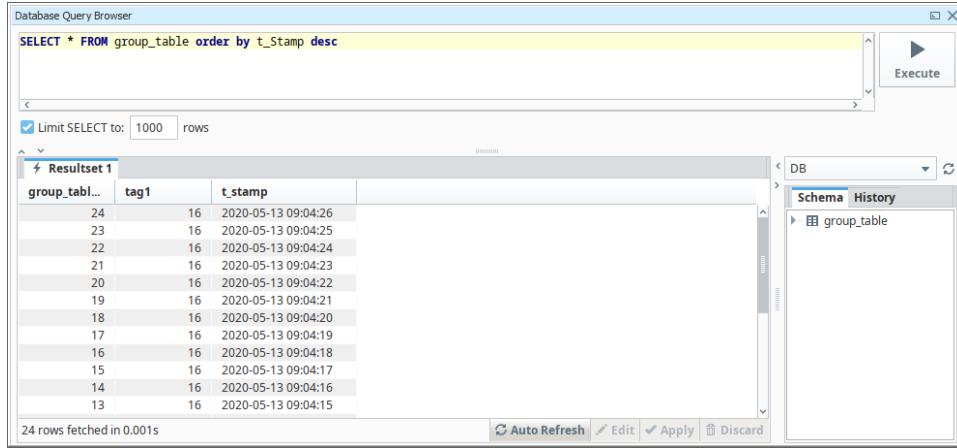
1. Create a **Standard Transaction Group** and from your Tag Browser, drag a single tag into your Transaction Group's Basic OPC /Group Items section. For this example, the tag is called 'tag1.'
2. Set the Update Mode on 'tag1' to **OPC to DB** and set its **Target Name** to be 'tag1.' The Target Name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

Your Transaction Group will look like the screenshot below:



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 through the [Database Query Browser](#) as shown below:



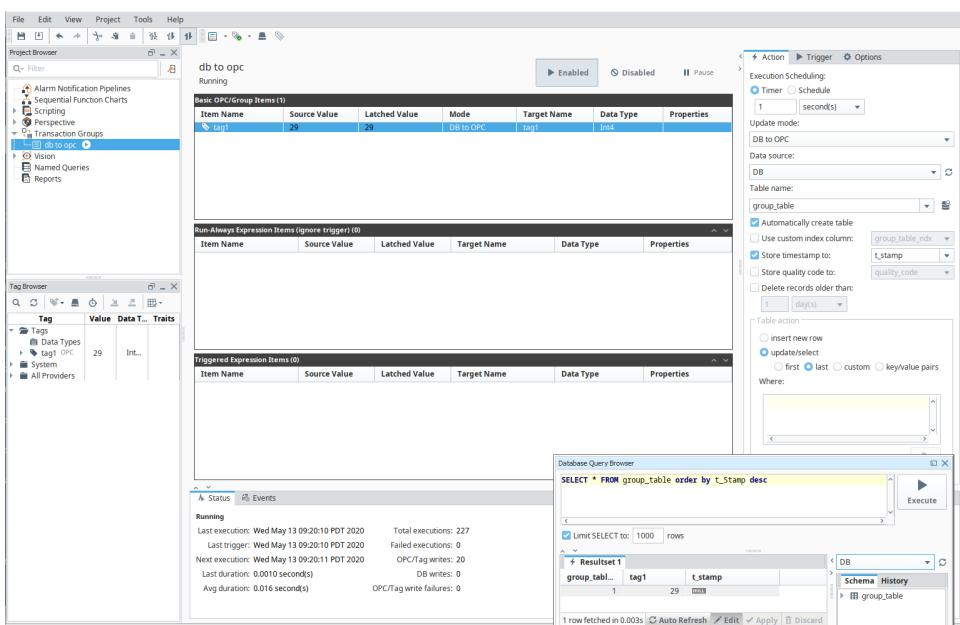
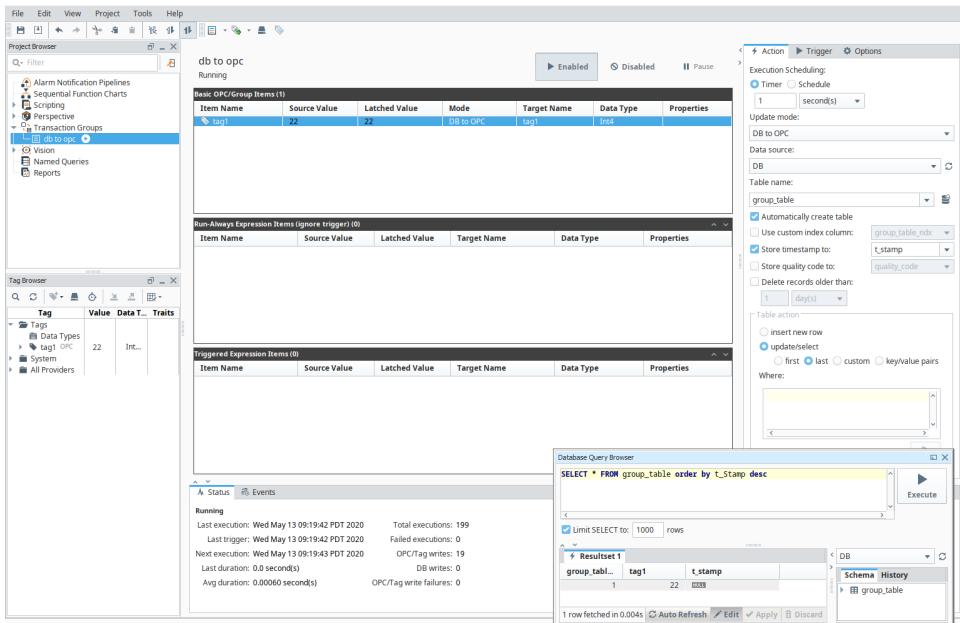
The screenshot shows the Database Query Browser interface. The query window at the top contains the SQL command: `SELECT * FROM group_table order by t_Stamp desc`. Below the query window, there is a checkbox for "Limit SELECT to: 1000 rows". The results are displayed in a table titled "Resultset 1" with two columns: "group\_table..." and "tag1". The "tag1" column contains the value 16. The "t\_stamp" column shows various dates and times from 2020-05-13 09:04:16 to 2020-05-13 09:04:26. To the right of the results, there is a schema browser window titled "DB" showing the "group\_table". The status bar at the bottom indicates "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.

From the screenshots below, 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.



## Bi-directional OPC Wins

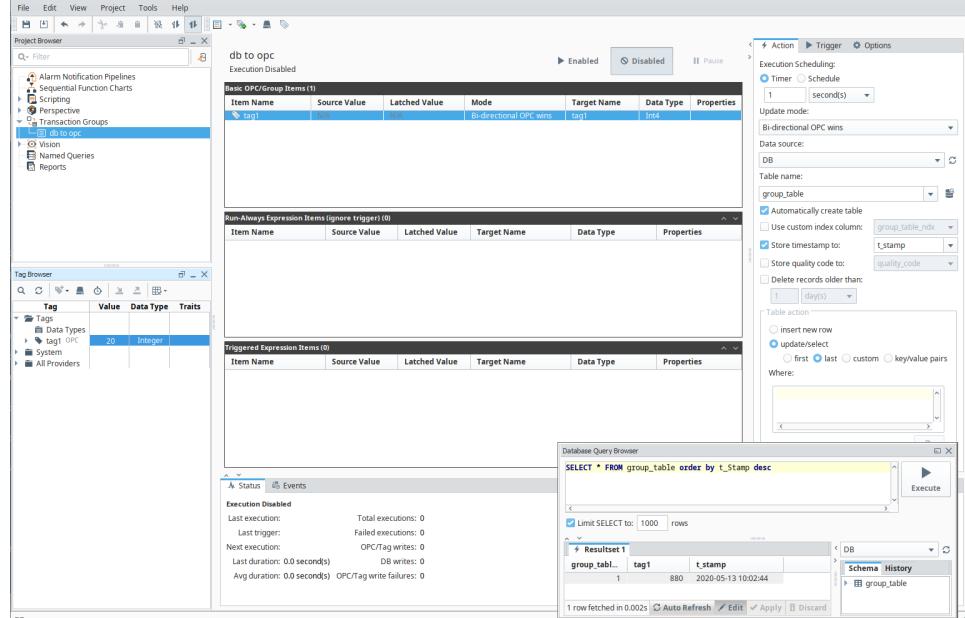
**Bi-directional OPC wins** means that Ignition will Read and Write to both the database and OPC Server. However, on initial group start, if the OPC and database values are different, the OPC value will win and the Transaction Group will write OPC values to the database.

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.

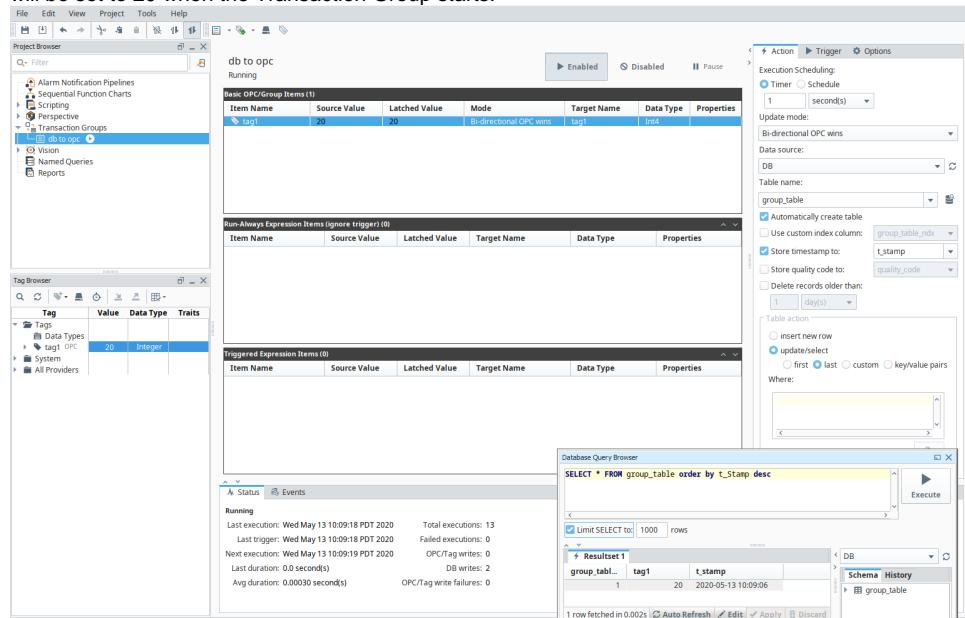
What you will have at this point is a bi-directionally controlled Transaction Group where any change to tag1's value will be reflected on the database and any change database side for the 'tag1' column value will be reflected on your 'tag1' tag.

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. This can be observed below:



Notice how the Transaction Group is disabled, 'tag1' value is 20, and the 'tag1' column value is 880. When the group is enabled, since the OPC and database values are different, the Update Mode being **'Bi-directional OPC wins'** means the 'tag1' column value will be set to 20 when the Transaction Group starts.



## Bi-directional DB Wins

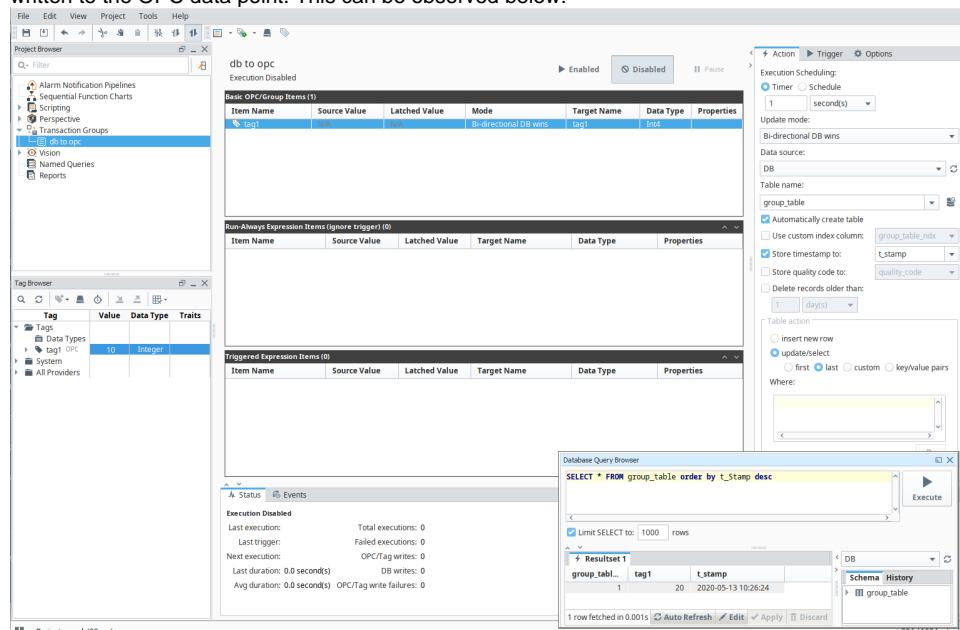
**Bi-directional DB wins** means that Ignition will Read and Write to both the database and OPC Server. However, on initial group start, if the OPC and database values are different, the database value will win and the Transaction Group will write database data to your OPC data points.

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.

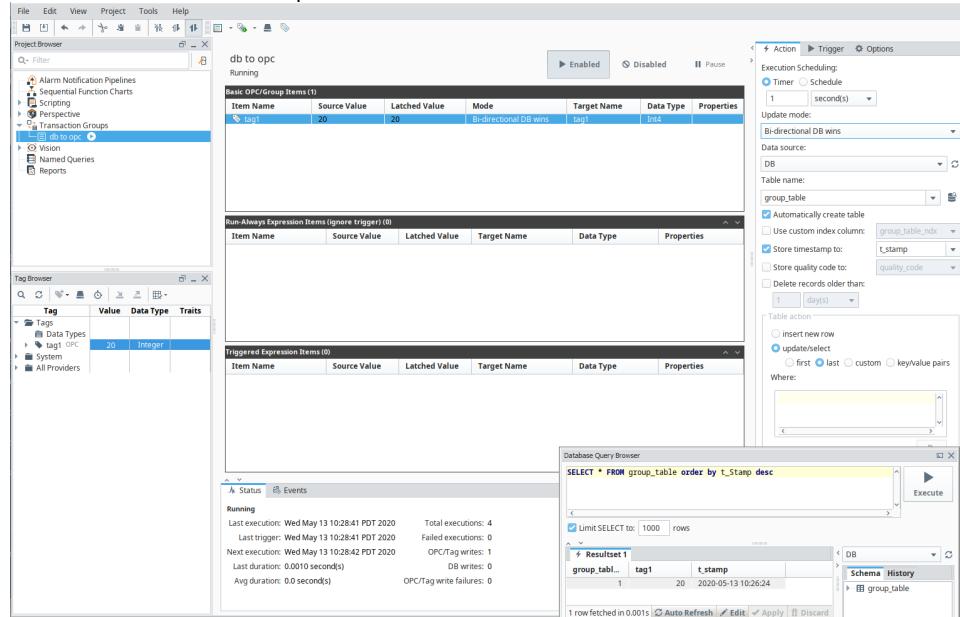
What you will have at this point is a bi-directionally controlled Transaction Group where any change to tag1's value will be reflected

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. This can be observed below:



Notice how the Transaction Group is disabled, 'tag1' value is 10, and the 'tag1' column value is 20. When I enable the group, since the OPC and database values are different, the Update Mode being '**Bi-directional DB wins**' means the tag1 tag value will be set to 20 when the Transaction Group starts.



## Related Topics ...

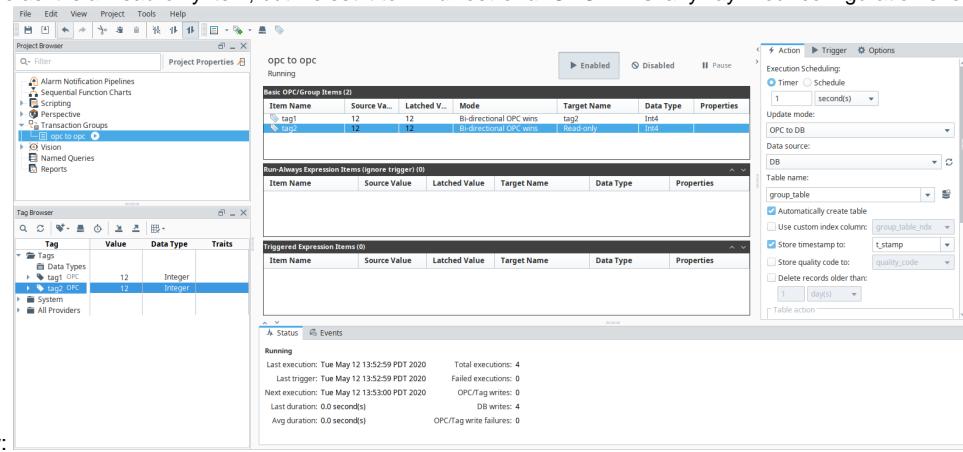
- Understanding Transaction Groups
- Types of Groups

# OPC to OPC Transaction Group

## Configuring Transaction Group for OPC to OPC Interaction

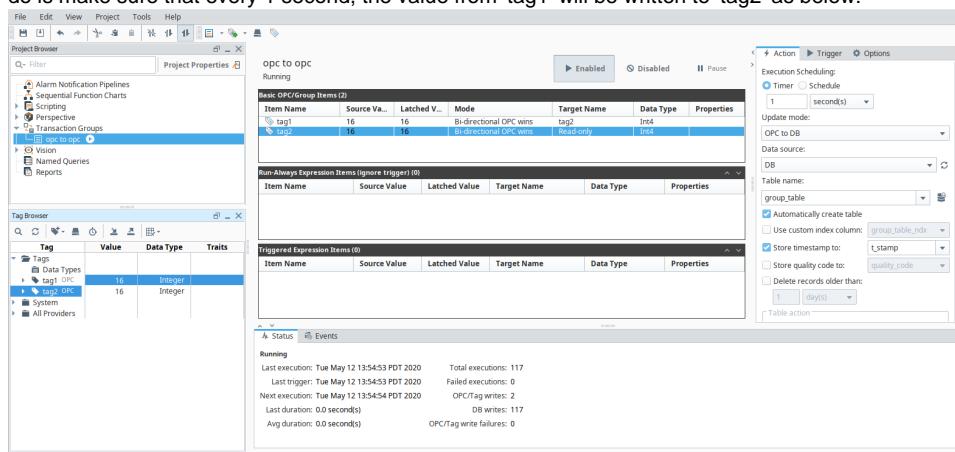
Transaction groups are generally used to channel OPC data to a database or vise-versa. It is also possible to configure your Standard Transaction Group to be able to get information from one OPC data point to another. This is useful in the event that you have tags coming from one PLC and you need the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

1. Create a **Standard Transaction Group** and from your Tag Browser, 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 PLC's.
2. Set the Mode on 'tag1' to **'Bi-directional OPC Wins'** and set its **Target Name** to be 'tag2.'
3. Set the Mode on 'tag2' to **'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. Your configuration should match what is



shown below:

do is make sure that every 1 second, the value from 'tag1' will be written to 'tag2' as below:



What this will