

Ignition 8.3 User Manual — Tag Historian and SQL Bridge Modules

Generated 2026-10-12 from
<https://www.docs.inductiveautomation.com/docs/8.3/>

Historian

The Historian Core module lets you log and retrieve tag history using a variety of storage options. When history is enabled on a tag, its values are stored through a configured historian provider. These providers include embedded options, external databases, and other storage types, each with different capabilities depending on your system's needs. For a full overview of available providers, see the [History Providers](#) page.

Tag history can be displayed in charts and tables, or accessed via bindings, reports, and scripting.

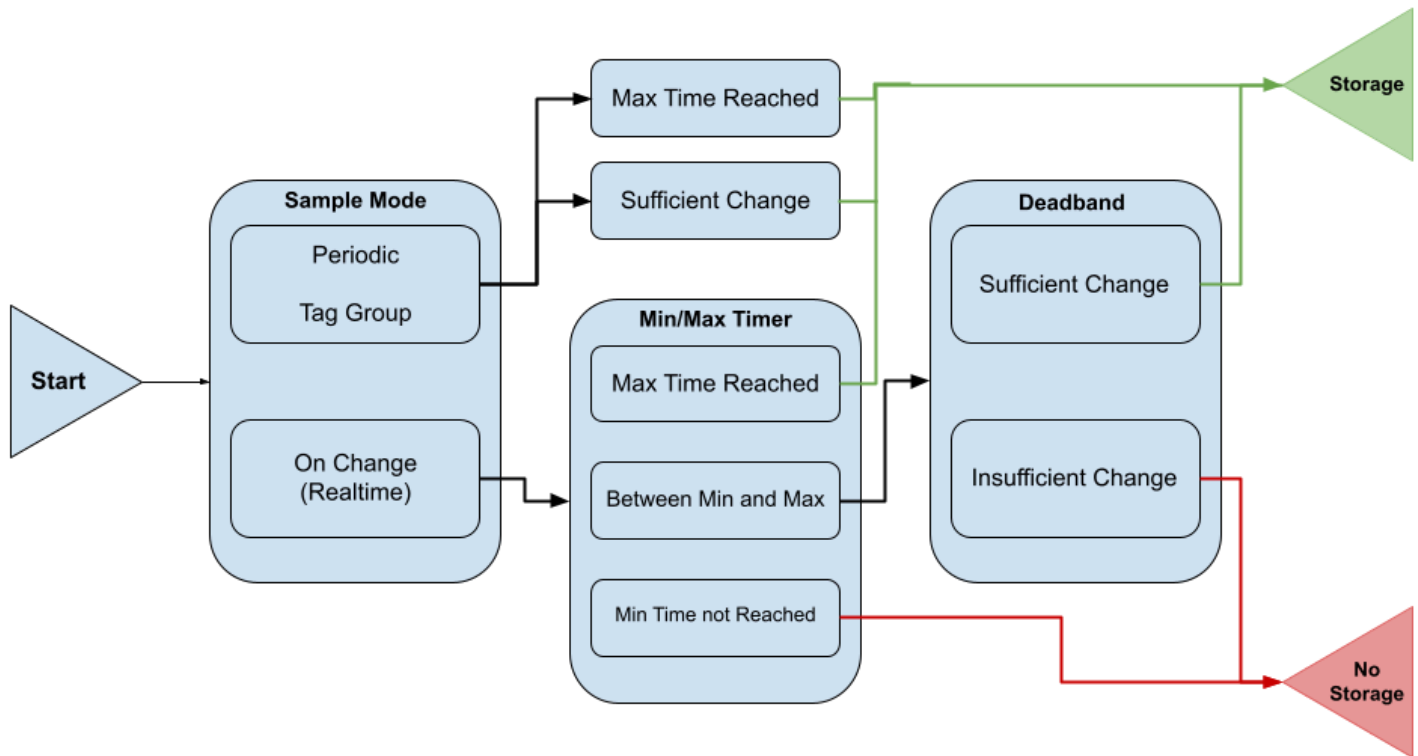
How the Historian System Works

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

Deciding When to Store

The system determines whether a sample should be stored based on sample mode, timer settings, and deadband evaluation.

When history is enabled on a tag, several checks occur. These are handled by components called actors. The key ones are sample mode, min/max timer, and deadband, which align with similarly named tag settings.



Sample mode

The system first detects value changes using the sample mode actor.

- Periodic and Tag Group modes evaluate storage at regular intervals.
- On Change mode triggers evaluation whenever the tag changes.
- On Change mode also includes a timer that ensures a sample is collected if the tag has stayed static longer than the configured maximum time.

Tags are evaluated together if they share the same tag provider, history provider, and rate. Tags with different sample rates are evaluated independently.

Min/Max timer

The min/max timer checks if the minimum or maximum time between samples has been met.

- If the minimum time hasn't elapsed, the new value is discarded.
- If the maximum time is exceeded, a sample is collected, even if the deadband is not exceeded.

If using a Tag Group mode, maximum time values on the Tag Group override those on the tag unless left at default.

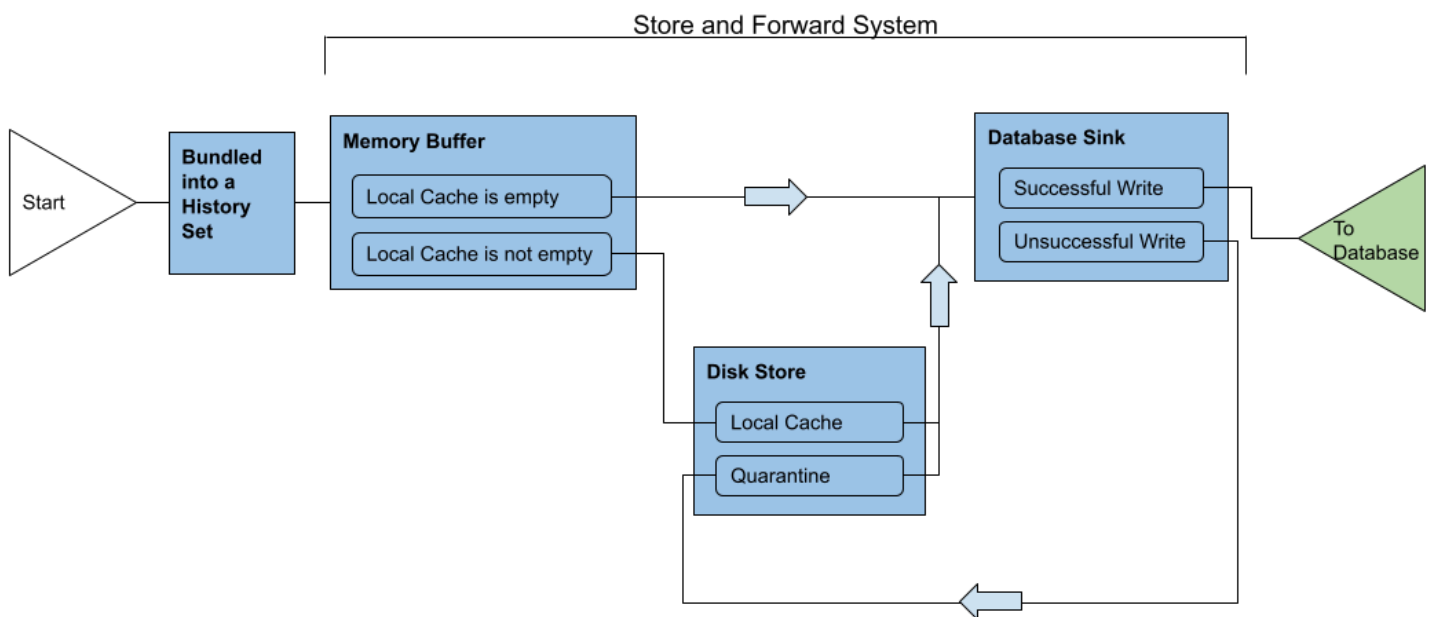
Deadband

The deadband actor compares the new value to the previous one. If the change exceeds the configured deadband, the system stores the new value.

Storing a Value

When a sample is collected, it's added to a history set. A history set contains a collection of samples from tags that share a history configuration. Periodic sample mode tags are grouped into a special "exempt" group.

The set is then sent to the **Store and Forward system** or written directly to disk, depending on the historian provider.



Memory buffer

When a history set reaches the memory buffer, it checks the current cache state. If no backlog is detected, the system immediately writes the data to the database sink. But if other sets are waiting to be written, the system routes this set to disk to wait its turn.

Disk store

The disk store holds records temporarily and separates them based on their condition:

- The local cache holds data that's ready to be written once the configured write time or size threshold is reached.
- The quarantine area stores records that couldn't be written successfully due to issues like database errors. These entries remain in quarantine until the user intervenes or the system retries.

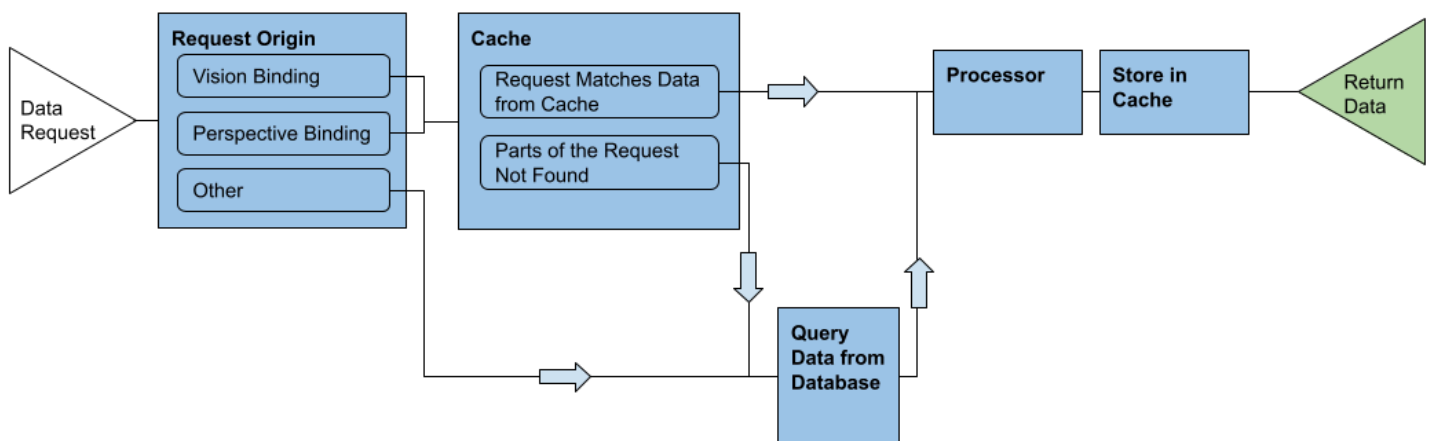
Database sink

The database sink is responsible for the final write to the database and performs the following tasks:

- Bundles records from the history set into a single transaction.
- Performs the insert using a prepared query.
- Caches tag identifiers in memory so repeated inserts don't require lookup queries.
- Retries failed records one at a time before sending them to quarantine.

Querying Historical Data

Historical values are stored in a database, but you should use Ignition's built-in tools to query them. These tools handle time alignment, interpolation, deduplication, and aggregation. The QuestDB-based historian supports native aggregation directly in the database engine.



You can query history using:

- Tag History bindings in **Perspective** and **Vision**
- Data sources in the **Report Data** tab of the Reporting module.
- Scripting functions under the **system.historian** namespace:

- `system.historian.queryValues`
- `system.historian.storeDataPoints`
- `system.historian.queryMetadata`
- `system.historian.storeAnnotations`

DEPRECATED

Custom Tag History Aggregates are deprecated and only work with the legacy `system.tag.queryTagHistory` and `system.tag.queryTagCalculations` functions, which are also deprecated. Although these features still work for older projects, they are not supported by the newer `system.historian` API and are not recommended for new development.

QUERY ONLY LICENSING

Users implementing a Scale Out architecture that will only query historical data on the front-end servers may be able to utilize a Query Only licensing option. This will limit the Historian Core module to read-only operations. Contact your Sales Representative for more information.

Request Origin

When data is requested, the system first identifies where the request came from. Queries from Perspective and Vision components often use caching to speed up repeated access. These requests are routed through a cache. Queries from scripting functions like `system.historian.queryValues` skip caching entirely and always pull directly from the database.

Cache

If the request supports caching, the system checks for matching data already stored in the cache. If all requested data is present, the system moves directly to processing. If the cache is missing values, the system retrieves just the missing data from the database.

Query Data from Database

The system prepares and executes one or more queries to gather the historical records. Since the Historian stores data in partitioned tables, this often involves checking which partitions fall within the requested time window.

If **pre-processed partitions** are enabled and the query qualifies, the system uses them instead of raw data tables. This significantly improves performance, but is not supported by the Core Historian.

Processor

The processor combines any cached and queried records into a unified dataset. It then slices the data based on the sample size and applies aggregation or interpolation where needed. For instance, if a query spans 10 minutes and requests 10 values, the processor creates ten 1-minute slices and evaluates the values in each.

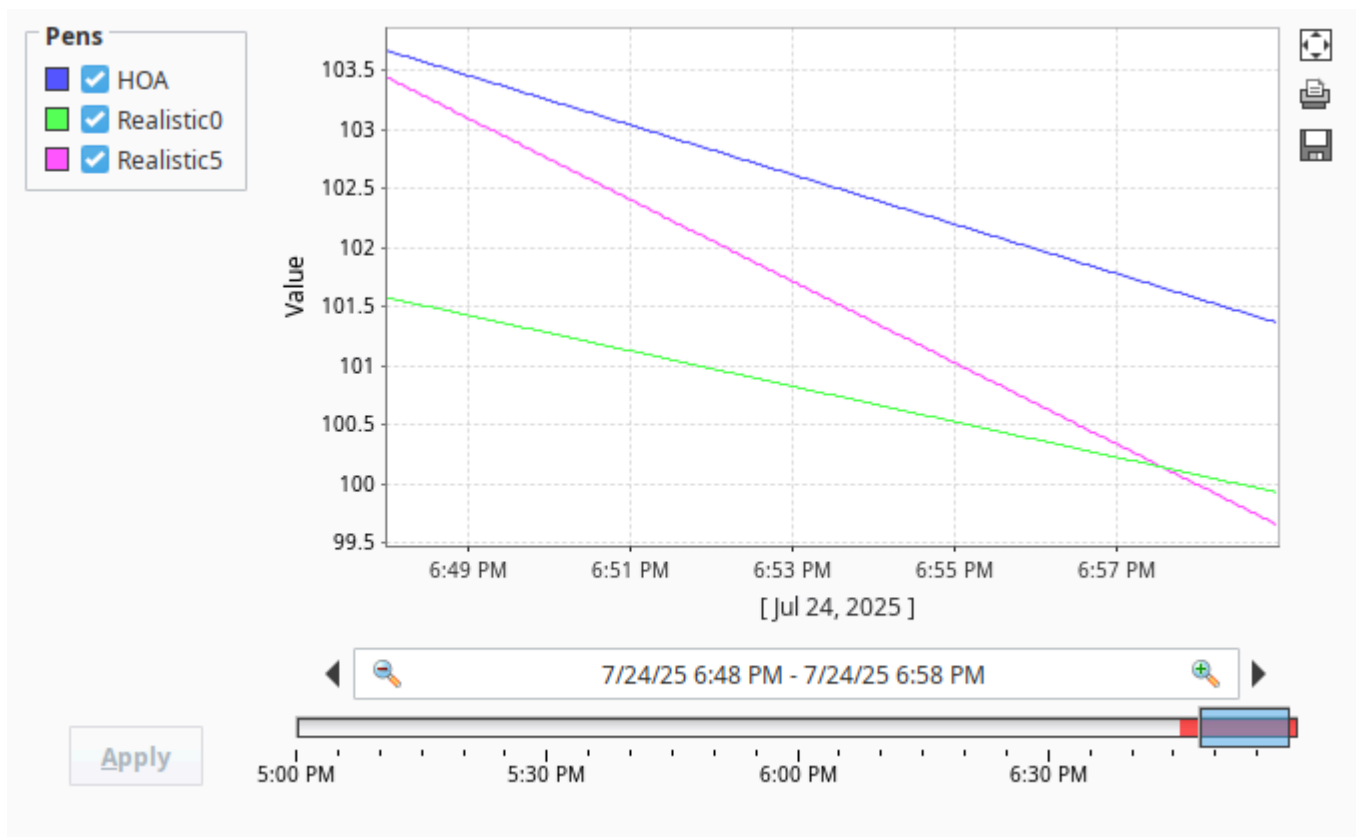
Store in Cache

If the query originated from a component that uses caching, the final result is stored in the cache after processing. Otherwise, the result is returned immediately to the requesting component or script.

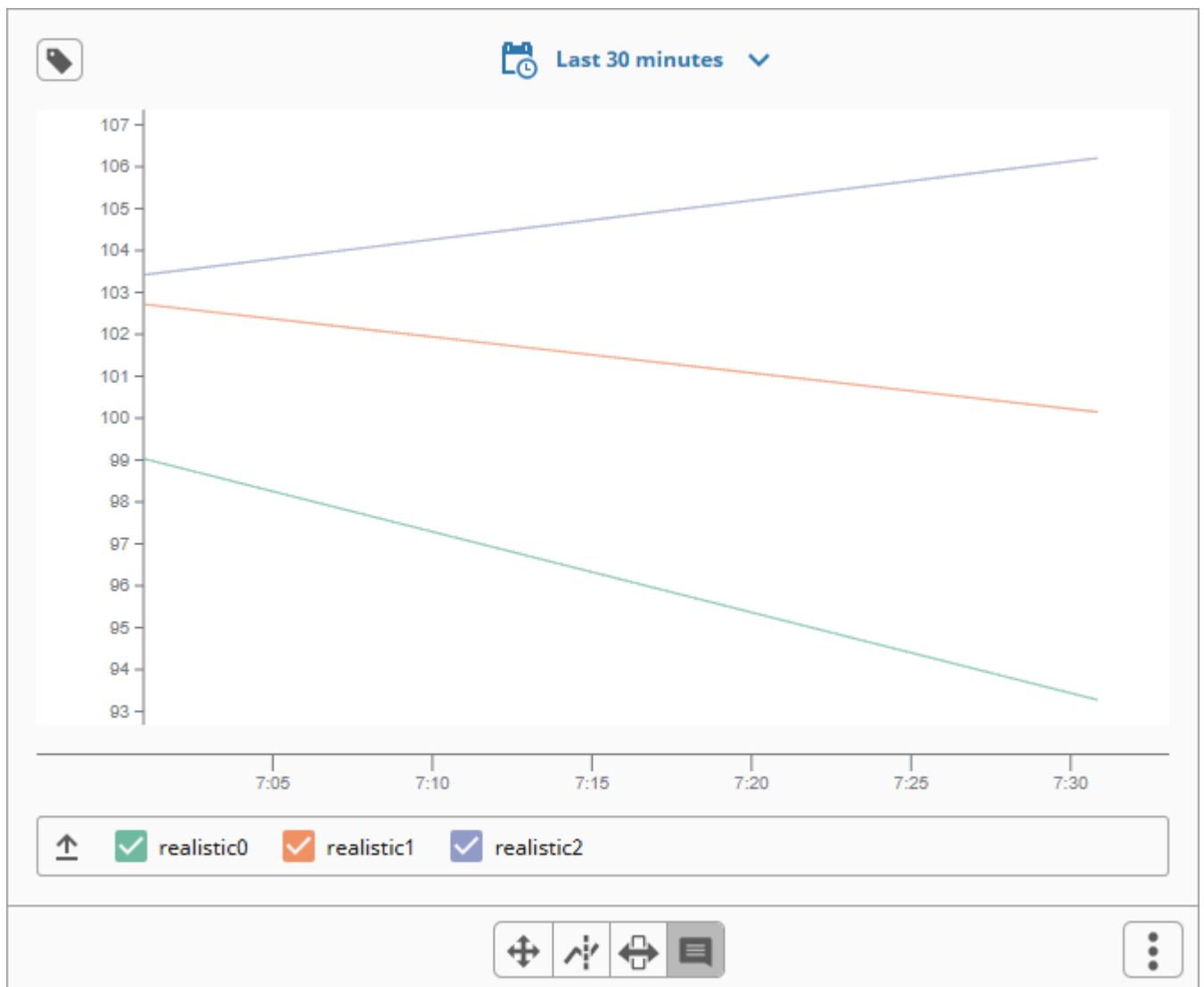
Display Historian Data

Both Vision and Perspective include Chart components designed for displaying historical tag data. Drag tags directly onto these charts to create pens. They support features like multiple axes, moving averages, subplots, and runtime controls for adjusting trends.

- **Easy Chart** in Vision



- **Power Chart** in Perspective



Store and Forward

The **Store and Forward** system stages historical data before writing it to the Internal Historian (Legacy) or external SQL historians. This buffering protects against data loss if the database is temporarily unavailable.

- The Internal Historian (Legacy) and external SQL providers always route historical data through Store and Forward. If the database cannot be reached, Store and Forward will temporarily queue records until the connection is restored.
- The Core Historian writes directly to disk using a write-ahead log (WAL) and does not use Store and Forward.

Data Storage

Each historical record includes the tag value, quality, and timestamp (to the millisecond). Whether a value is stored depends on the tag's sample mode, deadband settings, and min/max time

configuration.

For technical details on the tables used by the Historian, see the [Ignition Database Table Reference](#).

Configuring Tag History



INDUCTIVE
UNIVERSITY

Configuring Tag History

[Watch the video](#)

You can log historical data by enabling history on individual tags. Once enabled, tag values are stored using a configured historian provider. Depending on the provider, data may be routed through the Store and Forward system or written directly to disk.

When history is enabled, each record includes a millisecond-resolution timestamp, tag quality, and the original value. Values are stored based on the tag's history settings, such as deadband and sampling mode. Some providers also track execution metadata to help detect stale values.

Enable History on a Tag

History must be enabled on a tag in the Tag Editor before you can begin logging any values.

NOTE

Dataset-type tags are not supported by the historian system.

1. In the **Tag Browser**, select one or more tags.
2. Right-click the selection and choose **Edit Tag** .
3. In the **Tag Editor**, scroll to the **History** section or select it from the left panel.
4. Enable the **History Enabled** property.
5. Select a **Storage Provider**.
6. Set the **Sample Mode** to **Tag Group**.
7. Choose a **Historical Tag Group**, such as **Default Historical**.
8. Click **OK** to apply your changes.

Tags with history enabled display the **History**  icon in the Tag Browser.

Categories

All Properties

Basic
Meta Data
1a Value
Numeric
Security
Scripting
Alarms
History
Custom

History

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

OK
Apply
Cancel

NOTE

To record history for members of a UDT, edit the UDT definition and enable history on individual members. These settings will apply to all UDT instances.

Tag History Configuration

Sample Mode

This setting controls the frequency at which the tag history system evaluates **if** tag values should be stored.

- **On Change**: Stores a value when the tag's value changes.
- **Periodic**: Stores a value at fixed intervals based on the **Sample Rate**.
- **Tag Group**: Uses the execution rate of the selected **Historical Tag Group**.

Historical Tag Group

The Historical Tag Group setting is available when Sample Mode is set to Tag Group. It determines how often the historian checks the tag for changes that should be stored. This property uses a configured **Tag Group**, which defines an execution rate.

The evaluation frequency should align with or be slower than the tag's primary Tag Group to avoid redundant checks. For example, if the tag executes at a 1,000ms rate and the Historical Tag Group runs at 500ms, the historian will evaluate the tag twice per update. This can result in unnecessary processing and potential storage of duplicate values.

Choosing a Historical Tag Group with the right execution rate helps optimize performance and ensures storage integrity.

Max and Min Time Between Samples

These settings define how often values are stored when using On Change sampling. Normally, values are only stored when a change occurs that exceeds the deadband. If the tag value remains the same, no new records are inserted, regardless of how much time passes.

- **Max Time Between Samples** sets the maximum number of Tag Group executions that can pass before a value is forcibly stored, even if it hasn't changed. This ensures you don't go too long without a data point.
- **Min Time Between Samples** prevents values from being stored more frequently than a set interval, even if changes exceed the deadband.

NOTE

Max Time Between Samples must be at least 1000 milliseconds.

These settings are useful for maintaining consistent records in scenarios where tag values may be static for long periods or fluctuating too rapidly. For example, setting Max Time Between Samples to 5 and using a Tag Group that runs every 1 second will guarantee a record is stored at least every 5 seconds, regardless of change.

Use these options carefully to balance data integrity with database storage size.

Deadband Discrete Style and Analog Compression

This section describes the Deadband Style property options Auto, Analog, and Discrete because it may not be immediately clear how the deadband value is used differently depending on whether the tag is configured as a Discrete tag or as an Analog tag. Discrete values are more straightforward, registering a change any time the value moves +/- the specified amount from the last stored value. However, with Analog tags 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. The following examples show the process in action, comparing a raw value trend to a "compressed" trend.

Discrete

Storage: The deadband will be applied directly to the value. That is, a new value (V1) will only be stored when:

$$|V1 - V0| \geq \text{Deadband}$$

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

$$\text{Upper Slope} = ((\text{NewValue} + \text{Deadband}) - \text{PreviousValue}) / (\text{NewTimestamp} - \text{PreviousTimestamp})$$

$$\text{Lower Slope} = ((\text{NewValue} - \text{Deadband}) - \text{PreviousValue}) / (\text{NewTimestamp} - \text{PreviousTimestamp})$$

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

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

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

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

Interpolation: The value will be interpolated linearly between the last stored value and the next value. For example, if the value at Time0 was 1, and the value at Time2 is 3, selecting Time1 will return 2.

USING DEADBAND WITH THE CORE HISTORIAN

Out of order writes (such as with the Analog deadband style) for the Core Historian can be taxing on your system. To avoid potential impacts on performance and I/O utilization, it is recommended to use either the Discrete deadband style or turn deadband mode off and use the [Periodic Sample Mode](#).

Auto

This setting will automatically pick either Analog or Discrete, based on the data type of the tag. Auto is the default Deadband Style when initially configuring tag history.

- 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.

DEADBAND SETTINGS

In addition to the Deadband Style, your tag history settings also include the Deadband Mode and Historical Deadband properties.

- **Deadband Mode** is set to Absolute by default. If changed to Percent, the deadband is calculated as a percentage of the tag's engineering unit span.
- **Historical Deadband** applies specifically to historical data evaluation.
- If **Deadband Mode** is set to Off, all values are stored when the timestamp changes.

Interpolated Values

When querying historical data, Ignition may interpolate values to fill gaps where no raw data exists in a given time window. Interpolation helps maintain consistent charting, reporting, and aggregation behavior when data is sparse or unevenly distributed.

Interpolation only occurs when:

- The selected [Aggregation Mode](#) supports it
- **Avoid Interpolation** is disabled
- No raw values exist in a given time window

Interpolation Example

In the example below, a tag history query from 1:30pm to 5:30pm is executed using 60-minute intervals. The query returns three windows with data and one empty window.

Raw Data	
	t_stamp localatag1
Time Window 1	Nov 15, 2021 1:33 PM0.04
	Nov 15, 2021 1:36 PM5.6
	Nov 15, 2021 1:39 PM45.6
	Nov 15, 2021 1:42 PM3.45
	Nov 15, 2021 1:45 PM96.45
Time Window 3	Nov 15, 2021 3:33 PM9.68
	Nov 15, 2021 3:36 PM5.6
Time Window 4	Nov 15, 2021 4:33 PM30.04
	Nov 15, 2021 4:36 PM9.6
	Nov 15, 2021 4:39 PM23.6
	Nov 15, 2021 4:42 PM76.49
	Nov 15, 2021 4:45 PM2.54

- Time Window 1 (1:30–2:30): 0.04, 5.6, 45.6, 3.45, 96.45
- Time Window 2 (2:30–3:30): no data
- Time Window 3 (3:30–4:30): 9.68, 5.6
- Time Window 4 (4:30–5:30): 30.04, 9.6, 23.6, 76.49, 2.54

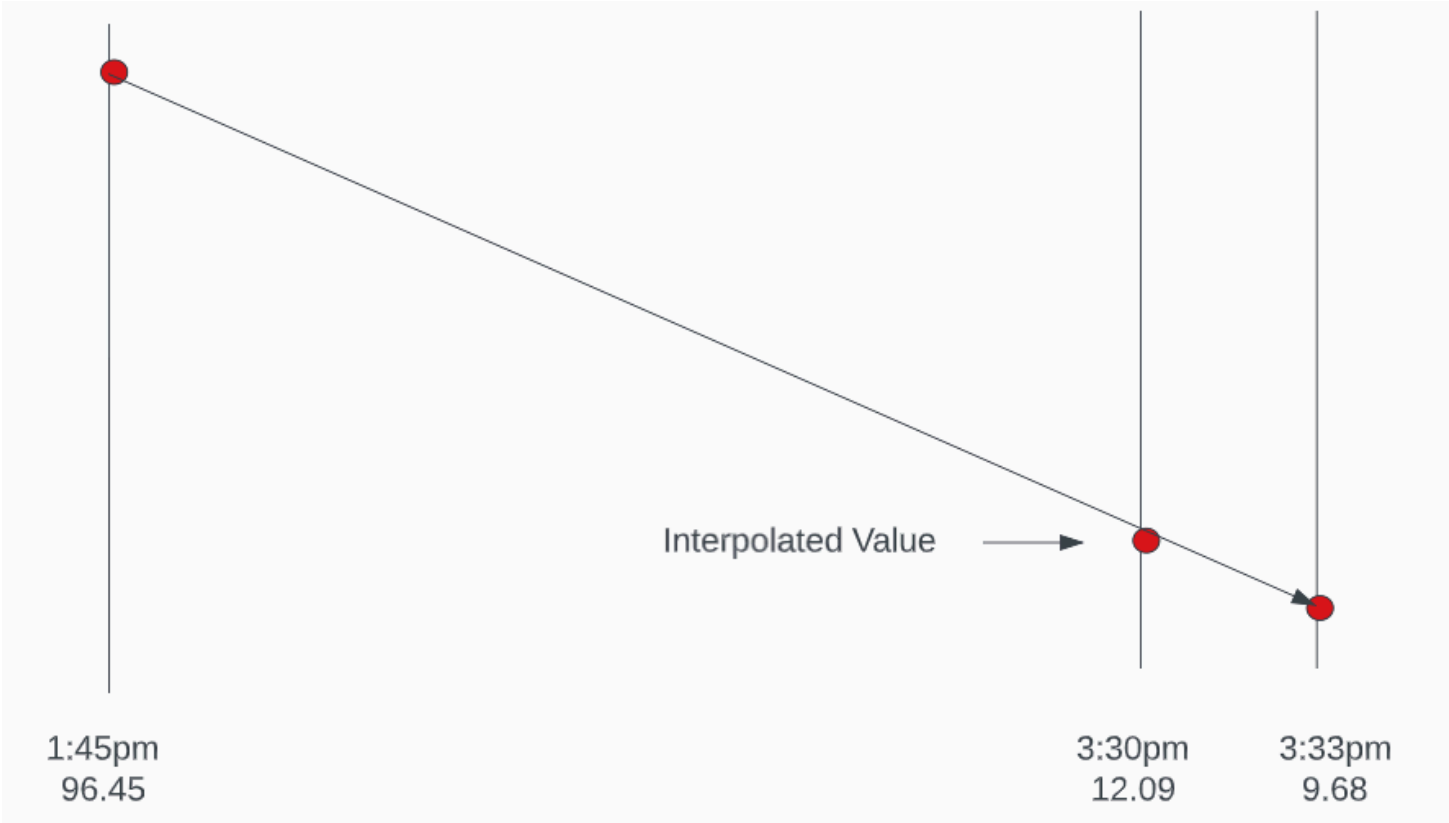
With **SimpleAverage** selected as the Aggregation Mode, the system averages the raw values in each window:

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

Since there are no values for Window 2, Ignition interpolates using the last value of Window 1 (96.45) and the first value of Window 3 (9.68):

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

The resulting linear interpolation at 3:30pm is **12.09**. This is calculated by drawing a line between the endpoints and marking the value at the missing timestamp:



Unless the **Average** Aggregation Mode is used, all other modes will use the same interpolated value of **12.09**.

Time-Weighted Average Interpolation

The **Average** Aggregation Mode uses a time-weighted formula instead of a simple arithmetic mean. This includes both raw and interpolated values with durations factored in.

Average

Queried Data

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

Time-weighted average formula:

Average = (0.5 × abs(currVal – prevVal) × timediff) + (timediff × min(currVal, prevVal))

Using the earlier example:

```
prevVal = 96.45
currVal = 60.29
timediff = 2,699,999 ms
```

$$(0.5 \times \text{abs}(60.29 - 96.45) \times 2,699,999) + (2,699,999 \times 60.29) = 211,606,751.63$$

NOTE

To calculate the interpolated value of 60.29 (used above), Ignition uses a **retValue** interpolation formula:

$$\text{retValue} = A + ((B - A) \times ((ts - A_{ts}) / (B_{ts} - A_{ts})))$$

Where:

- $A = 96.45$
- $B = 9.68$
- $B_{ts} - A_{ts} = 3:33\text{pm} - 1:45\text{pm} = 108 \text{ minutes}$
- $ts - A_{ts} = 2:30\text{pm} - 1:45\text{pm} = 45 \text{ minutes}$

$$96.45 + ((9.68 - 96.45) \times (45 / 108)) = \mathbf{60.29}$$

Preceding Window Calculation

To include raw time-weighted values from 1:33pm to 1:45pm:

NOTE

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

Sub-intervals:

- 1:33 to 1:36:
 $(0.5 \times \text{abs}(5.6 - 0.04) \times 179,999) + (179,999 \times 0.04) = 507,597.18$
- 1:36 to 1:39:
 $(0.5 \times \text{abs}(45.6 - 5.6) \times 179,999) + (179,999 \times 5.6) = 4,607,974.4$
- 1:39 to 1:42:
 $(0.5 \times \text{abs}(3.45 - 45.6) \times 179,999) + (179,999 \times 3.45) = 4,414,475.47$
- 1:42 to 1:45:
 $(0.5 \times \text{abs}(96.45 - 3.45) \times 179,999) + (179,999 \times 3.45) = 8,990,950.05$

Subtotal = 18,520,997.10

Add the two totals:

$(211,606,751.63 + 18,520,997.10) / 3,419,999 = \mathbf{67.29}$

Interpolated Average for Empty Window

Now using **Average** mode for the empty window (2:30–3:30):

prevVal = 60.29

currVal = 12.09

timediff = 3,599,999 ms

$(0.5 \times \text{abs}(12.09 - 60.29) \times 3,599,999) + (3,599,999 \times 12.09) = 130,283,963.81$

Divide by window duration:

$130,283,963.81 / 3,599,999 = \mathbf{36.19}$

Seeded Values

Seeded values (also called boundary values) are additional values retrieved just outside the query range—one before the start time and one after the end time. These values support interpolation and certain aggregation modes, especially when data is sparse at the query boundaries.

- The **pre-query seed** appears just before the query's start time and is usually the first row in the result set.
- The **post-query seed** appears just after the query's end time and is often used to complete an interpolation window.

These boundary values are automatically included unless the query is configured to exclude them. To omit seeded values, you must meet all of the following:

- Interpolation must be disabled (either by tag mode or query configuration)
- **Deadband Style** must be set to **Discrete**
- The query must include `noInterpolation=True`
- The query must include `includeBoundingValues=False`

Seeded values are not returned when using **raw data queries**, except in one special case using scripting.

Pre-query Seed Value

The pre-query seed is a value retrieved immediately before the start time of the query. This value is used for interpolation and often appears as the first result in the dataset. When interpolation is enabled and the result set is not raw, this value is included by default.

An exception exists when using `system.historian.queryRawPoints` with `returnSize=-1` (for raw data) and `includeBoundingValues=True`. In that case, the pre-query seed is still returned to allow proper interpolation on the client side.

Post-query Seed Value

The post-query seed is a value recorded just after the query end time. It is included when interpolation is enabled and helps calculate the last interpolated value for the final time window.

Seeded post-query values are not returned if the tag uses a **Discrete** deadband style. Tags with **Analog** deadbands or other interpolation-friendly settings will include this value.

When querying for local tags and the query range ends near or at the present time, Ignition may use the current tag value as the post-query seed, rather than fetching the last database record. This improves responsiveness when charting live data.

NOTE

If a value appears outside the requested range, it is usually a seeded value. This often occurs when:

- The query window is very short and contains no data
- The query window ends in the future and no values exist yet

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 Mode on Vision Tag History bindings to **On Change**
- Setting the **returnSize** parameter on [system.historian.queryRawPoints](#) or [system.historian.queryAggregatedPoints](#) to -1.

NOTE

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

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

History Providers

The Historian Core module supports multiple storage backends through a system of history providers. Each provider manages its own storage configuration and database schema, allowing you to log tag history in different ways depending on your system's needs. You can configure and manage history providers on the **Services > Historians > Historians** page on the Gateway.

Providers range from lightweight embedded engines to external SQL databases and remote Gateways. This flexibility lets you balance performance, scalability, and redundancy. Each provider type has unique behaviors, storage formats, and limitations. For technical details on the underlying storage schemas, see the [Ignition Database Table Reference](#).

For automation or custom tooling, you can use the `system.historian` scripting functions to interact with history providers programmatically.

Creating a History Provider

Most provider types must be created manually. To create a new provider, complete the steps below.

1. Go to **Services > Historians > Historians** on the Gateway.
2. Click **Create Historian**.
3. Select a historian type and click **Next**.
4. Enter a name and configure the settings.
5. Click **Create Historian** to finish.

CSV Historian

The CSV Historian imports historical data from a CSV file. This provider is read-only and does not support realtime tag history storage.

Settings

Property	Description
Name	Specifies the unique name used to identify the historian provider.
Description	Provides an optional user-defined description for the provider.
Enabled	Set whether this historian is enabled or disabled within the Gateway.
File Path	Specifies the CSV file path. Required format: timestamp column followed by value columns, with tag names defined in the header row.
Relative Timestamps	When enabled, treats the first timestamp as a reference point, interpreting subsequent timestamps as relative offsets from this initial time.

Historian Splitter

The Historian Splitter writes tag history to two configured providers simultaneously. This configuration is useful for redundancy, system migrations, or archiving data in multiple locations. When querying history, results are typically returned from the first provider, unless limits are configured.

Settings

Main

Property	Description
Name	Specifies the unique name used to identify the historian provider.
Description	Provides an optional user-defined description for the provider.
Enabled	Set whether this historian is enabled or disabled within the Gateway.

Storage

Property	Description
First Connection	Data is stored to both connections equally. However, all queries execute against the first connection, unless a limit is imposed using the settings below, or the first connection is unavailable.
Second Connection	Specifies the secondary history provider. Data is written here alongside the first connection, but this provider is not queried unless query limits are applied or the first connection fails.

Querying

Property	Description
Limit First Connection Query	If enabled, restricts query operations to a specific time window on the first provider.
Limit Length	Defines the size of the time window applied when query limits are enabled.
Limit Units	Specifies the time units for the limit window (e.g., minutes, hours, days).

DB Table Historian

The DB Table Historian enables Ignition to read historical data from external or custom-managed SQL tables using a flexible path syntax. This type is read-only and is commonly used for integrating third party or legacy historical datasets into Ignition’s trending and reporting tools.

Because this provider requires mapping raw SQL tables into a format Ignition can interpret as tag history, setup is more involved than with other historian types. You'll need to configure column mappings, timestamp fields, and optional filters using specialized historian paths.

See the [DB Table Historian Provider](#) page for details.

Core Historian

The Core Historian is an embedded, high-throughput time-series database using QuestDB. The Core Historian supports partitioning, deduplication, archiving, and native aggregation.

NOTE

The Core Historian writes directly to disk using a high-performance write-ahead log (WAL) system. It does **not** use Store and Forward, so it bypasses the buffering and retry mechanisms used by other historian types.

See the [Core Historian](#) page for more details.

Internal Historian (Legacy)

The Internal Historian (Legacy) is a lightweight embedded historian that uses a local SQLite database. It is well-suited for Edge devices, standalone installations, or small systems that require basic history storage without the overhead of an external database. This provider supports automatic data pruning and remote synchronization to another Gateway.

NOTE

SQLite-based historians do not record scan class execution data.

Settings

Main

Property	Description
Name	Specifies the unique name used to identify the historian provider.
Description	Provides an optional user-defined description for the provider.
Enabled	Set whether this historian is enabled or disabled within the Gateway.

Limits

Property	Description
Time Limit Enabled	If enabled, old data will be pruned based on its age.
Time Limit Size	Specifies how long data should be retained before it is pruned.
Time Limit Units	Sets the time unit (e.g., Days, Weeks) used for the time-based limit.
Point Limit Enabled	If enabled, data will be pruned once a specified number of records is reached.
Point Limit Size	Defines the maximum number of data points the historian will store.

Sync Settings

Property	Description
Remote Sync Enabled	Enables the provider to sync history data to a remote Gateway.
Remote Gateway Name	The Gateway to target for remote synchronization. The Historian Core module must be installed, and 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.
Sync Frequency Units	Sets the unit of time (e.g., seconds, minutes) used for sync frequency.
Max Batch Size	The maximum number of data points that will be sent per batch to the remote Gateway.
Enable Schedule	If enabled, the data will only be synchronized during the times specified by the pattern provided.
Schedule	A comma separated list of time ranges (e.g., 9:00-15:00).

Metadata Storage

When enabled on a tag, metadata is stored automatically in system-managed tables. This includes properties like tag path, data type, units, and scaling. Metadata is preserved even if the tag is removed.

Metadata must be explicitly enabled in the Tag Editor by setting **Include Metadata** to true. This setting is found in the **History** section when editing a tag in the Designer.

Metadata storage is only available for the **Core** Historian and **Internal** Historian (Legacy).

OPC HDA Historian

The OPC HDA Historian connects to OPC HDA (Historical Data Access) servers to read historical tag data. This provider is supported only on Windows systems and requires the OPC COM module to function. It is a **read-only** provider and cannot store or forward tag history.

 NOTE

This provider cannot write tag history.

Settings

Main

Property	Description
Name	Specifies the unique name used to identify the historian provider.
Description	Provides an optional user-defined description for the provider.
Enabled	Set whether this historian is enabled or disabled within the Gateway.

Server

Property	Description
Connection Type	Select either a local or remote connection type.
Provider	Each OPC server is identified by a "progid". This is the name that will be used to connect to the server.
Use Flat Browsing	Flat browsing returns all items at once. This is less efficient than normal browsing, but if a server only supports flat browsing, select this option.

Remote Historian

The Remote Historian provider connects to another Ignition Gateway over the Gateway Network to read from or write to a remote history provider. This allows distributed storage or centralized access to

tag history across Gateways.

Settings

Main

Property	Description
Name	Specifies the unique name used to identify the historian provider.
Description	Provides an optional user-defined description for the provider.
Enabled	Set whether this historian is enabled or disabled within the Gateway.

Remote Gateway

Property	Description
Gateway	Specifies the name of the remote Gateway to connect to.
Provider	Specifies the name of the history provider on the remote Gateway.

Storage

Property	Description
Storage Allowed	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.
Max Group 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)

Simulator Historian

The Simulator Historian generates simulated tag history data for use in testing, training, and demonstration environments. It allows you to visualize trends, test charts and bindings, and

experiment with history queries all without needing a live system or real tag activity.

See the [Simulator Historian](#) page for more info.

SQL Historian

The SQL Historian stores tag history in an external SQL database such as MySQL, SQL Server, or PostgreSQL. It supports advanced features like partitioning for performance, pruning for data retention, stale value detection, and summary table generation through pre-processed partitions. This provider is well-suited for systems that require robust reporting, integration with external tools, or long-term historical data storage.

NOTE

The SQL Historian module is required in addition to the Historian Core module for this history provider to be listed on the Gateway.

Settings

Main

Property	Description
Name	Specifies the unique name used to identify the historian provider.
Description	Provides an optional user-defined description for the provider.
Enabled	Set whether this historian is enabled or disabled within the Gateway.
Data Source	Identifies the database connection used to store historical data.

Data Partitioning

Property	Description
Enable Partitioning	Enables time-based table partitioning to segment historical data. Partitioning improves query performance and simplifies data lifecycle management.
Partition Length	Specifies the duration of each partition (e.g., 1, 7, or 30).
Partition Units	Defines the unit of time for each partition (e.g., Days, Weeks, Months).
Number of past partitions queried for seed values	New in 8.3.8 Controls how many partitions prior to the query window are searched when finding a seed value . A value of 0 will globally disable seed queries, and a negative number will search every available partition.
Enable Pre-processed Partitions	Pre-processed partitions will use more space in the database, but can improve query speed by summoning data, reducing the amount that must be loaded.
Pre-processed Window Size (seconds)	When the pre-processed partitions property is enabled, this setting defines the window size. Configuring this setting has a direct impact on whether or not the pre-processed data will be used for tag history queries. If the query window size is greater than or equal to the pre-processed window size, then the pre-processed tables are used instead of the raw data tables. If the query window size is less than the pre-processed window size, then the raw data table is used. Default is 60 seconds.

Seed Values

A seed value is the last recorded value found before the start of a query window. It is used to establish a starting point for the query so that a value can be returned at the beginning of the requested time range, even when no data was logged at that exact time.

By default, if a partition contains no data, the historian searches one partition back for a seed value and returns a 0 value if none is found. When data is logged infrequently or partition sizes are small, a seed value may not exist in the partitions searched, resulting in a misleading 0. This is especially noticeable when a Historian Splitter points to two databases with different partition sizes, where the same query may return a real value from one database but a 0 from the other. Use the **Number of**

past partitions queried for seed values setting to control how many previous partitions to include in the search, so that consistent seed values are returned regardless of partition size.

Data Pruning

Data pruning works by deleting old partitions. Therefore, data will only be removed when a partition has no data younger than the prune age.

ENABLE PRUNING ON ONE HISTORIAN PER DATABASE

If you create multiple SQL Historians that use the same database, make sure to only enable data pruning on one of them. When more than one historian pointing to the same database has pruning enabled, their pruning tasks conflict and the data sink enters a critical failure state. [Store and forward](#) then stops processing data, and the pending count and quarantined data grow indefinitely until pruning is disabled on all but one historian.

Property	Description
Enable Data Pruning	Data will be deleted after the specified time period.
Prune Age	Specifies the maximum age of data to retain in the database.
Prune Age Units	Defines the units of time used to interpret the prune age.

Advanced

Property	Description
Enable Stale Data Detection	If enabled, tracks tag group executions to determine the difference between unchanging values, and values that are flat due to the system not running.
Stale Detection Multiplier	The multiplier for tag group rate used to determine when values are stale. If tag group execution is not recorded within this amount of time, values will be considered bad on query.

Pre-Processed Partitions

Pre-processed partitions are used to aggregate data partitions into smaller, more efficient partition tables. When enabled, the system will use the pre-processed window size (in seconds) and

summarize the raw historical data into blocks of this size. For example, if a tag was logged to the database every second, a window size of 60 seconds will summarize the 60 records logged into a single, much smaller set of records in the pre-processed table. On large systems with millions of historical records per month, this summarization of data can result in much smaller data partition tables that are more efficient to query.

When SQL-based history providers opt into pre-processed partitions, records stored by the provider are stored in two different sets of partitions: the normal partitions and a summarized pre-processed partition. In other words, a new partition table is created to store the pre-processed records. The pre-processed partitions use the following naming convention: `sqlth_data_X_YYYY_MM_windowSize`, where X is the provider driver ID from the `sqlth_drv` table. While this takes up more space in the database, it can improve query speeds by dramatically reducing the amount of resulting data points.

Pre-processed partitions are only used under specific conditions:

- When querying through legacy scripting functions such as [system.tag.queryTagHistory](#) and [system.tag.queryTagCalculations](#).
- When the requested query interval exceeds the configured Pre-processed Window Size.

For example, if the **Pre-processed Window Size** is set to 60 seconds, and a legacy query requests `intervalSeconds=90`, the query engine will retrieve data from the pre-processed partitions instead of the raw history tables, resulting in faster queries and smaller result sets. If the query uses a smaller interval (e.g., 30 seconds), the raw data tables are used instead.

CAUTION

The [system.historian.queryRawPoints](#) function does **not** leverage pre-processed partitions. Queries made through this function always access raw history data, even if pre-processing is enabled.

The Resultset 1 table below shows the normal partitions of 50 raw values between a highlighted time block spanning from 12:12:03.52 to 12:13:03.52.

Resultset 1 ×		Resultset 2 ×	
tagid	intvalue	DATE_ADD('1969-12-31 17:00:00', ...)	
1	-69	2022-05-18 12:13:04.042000	
1	-71	2022-05-18 12:13:03.040000	
1	-73	2022-05-18 12:13:02.039000	
1	-76	2022-05-18 12:13:01.038000	
1	-78	2022-05-18 12:13:00.037000	
1	-79	2022-05-18 12:12:59.035000	
1	-81	2022-05-18 12:12:58.035000	
1	-83	2022-05-18 12:12:57.034000	
1	-85	2022-05-18 12:12:56.032000	
1	-86	2022-05-18 12:12:55.030000	
1	-88	2022-05-18 12:12:54.028000	
1	-89	2022-05-18 12:12:53.027000	
1	-91	2022-05-18 12:12:52.026000	
1	-92	2022-05-18 12:12:51.025000	
1	-93	2022-05-18 12:12:50.024000	
1	-94	2022-05-18 12:12:49.022000	
1	-95	2022-05-18 12:12:48.021000	
1	-96	2022-05-18 12:12:47.020000	
1	-97	2022-05-18 12:12:46.020000	
1	-98	2022-05-18 12:12:44.017000	
1	-99	2022-05-18 12:12:42.015000	
1	-100	2022-05-18 12:12:36.009000	
1	-99	2022-05-18 12:12:33.005000	
1	-98	2022-05-18 12:12:32.004000	
1	-97	2022-05-18 12:12:30.001000	
1	-96	2022-05-18 12:12:28.999000	
1	-95	2022-05-18 12:12:27.998000	
1	-94	2022-05-18 12:12:26.998000	
1	-93	2022-05-18 12:12:25.997000	
1	-91	2022-05-18 12:12:24.996000	
1	-90	2022-05-18 12:12:23.994000	
1	-89	2022-05-18 12:12:22.994000	
1	-87	2022-05-18 12:12:21.993000	
1	-86	2022-05-18 12:12:20.992000	
1	-84	2022-05-18 12:12:19.991000	
1	-82	2022-05-18 12:12:18.990000	
1	-80	2022-05-18 12:12:17.990000	
1	-78	2022-05-18 12:12:16.990000	
1	-76	2022-05-18 12:12:15.988000	
1	-74	2022-05-18 12:12:14.986000	
1	-72	2022-05-18 12:12:13.985000	
1	-70	2022-05-18 12:12:12.984000	
1	-68	2022-05-18 12:12:11.983000	
1	-65	2022-05-18 12:12:10.982000	
1	-63	2022-05-18 12:12:09.982000	
1	-61	2022-05-18 12:12:08.981000	
1	-58	2022-05-18 12:12:07.980000	
1	-55	2022-05-18 12:12:06.979000	
1	-53	2022-05-18 12:12:05.977000	
1	-50	2022-05-18 12:12:04.976000	
1	-47	2022-05-18 12:12:03.974000	

With pre-processed partitions enabled, these 50 rows of data were also summarized into four rows shown on the Resultset2 tab displaying the pre-processed partition values.

Resultset 1 ×		Resultset 2 ×	
tagid	intvalue	vtype	DATE_ADD('1969-12-31 17:00:00', ...)
1	89	6	2022-05-18 12:13:03.525000
1	13	0	2022-05-18 12:13:03.525000
1	-47	98	2022-05-18 12:12:03.525000
1	-71	4	2022-05-18 12:12:03.525000
1	-100	1	2022-05-18 12:12:03.525000
1	-86	0	2022-05-18 12:12:03.525000
1	98	98	2022-05-18 12:11:03.525000
1	-45	5	2022-05-18 12:11:03.525000

You may notice that the pre-processed partition table also includes a `vtype` column. The `vtype` represents metadata about the record using the following flags:

- 0: Time-weighted average
- 1: Minimum value
- 2: Maximum value
- 4: Exit value (last in window)
- 32: First value flag (used with min/max to indicate which came first)
- 64: Entry value (first in window)

Each record in the set represents a different value for that 1 minute time window, each value given a different `vtype`. The first record above has a `vtype` of 98, which means that this record represents the entry (64), the first value (32), and the max (2) of the records between 12:12:03.52 and 12:13:03.52. The second record value is 4. This value represents the exit value for this same 1-minute time block. The third record value is 1, which represents the min value. Finally, the last record value is 0, which represents the direct average for the original 50 rows.

For more details on table structure, see the [Ignition Database Table Reference](#).

DB Table Historian Provider



INDUCTIVE
UNIVERSITY

DB Table Historian

[Watch the video](#)

The DB Table Historian Provider acts as a bridge between tables in a database connection and the Historian module, mapping columns in the tables as available tags, allowing historical tag queries to access the contents of the table. Usually historical tag queries can interact only with tables created by the Historian. Thus, a table created with a Transaction Group, or by some other means (manually creating the database tables, tables generated by third party systems, etc) couldn't be accessed via things like a Tag History binding. The DB Table Historian Provider solves this problem.

Note that utilizing the DB Table Historian Provider requires a valid database connection.

Configuring a DB Table Historian Provider

1. On the Gateway, go to **Services > Historians > Historians**.
2. Click **Create Historian**.
3. Select **DB Table Historian**, then click **Next**.

Create Historian

Create Historian

1. Select Type

2. Configure Historian

☐ CSV Historian

A historian implementation that provides historical data from a CSV file.

☐ Data Splitter Historian

Writes historical data simultaneously to two historians while managing query routing between them.

☒ DB Table Historian

Adds historian style query support to standard wide database tables.

☐ Internal Historian - QuestDB

A built-in local historian optimized for high-throughput, demanding workloads. Recommended choice for most use cases that require local timeseries data storage.

☐ Internal Historian - SQLite

A built-in local historian for lightweight applications.

☐ OPC HDA Historian

A historian that uses OPC-HDA to retrieve values from a third party historian.

☐ Remote Historian

Stores historical data to a remote historian using the Gateway Network.

☐ Simulator Historian

Provides simulation data for testing without an external data source.

☐ SQL Historian

A historian that leverages a SQL database for the storage and retrieval of historical data.

Cancel

Next

4. Enter a name for the provider.

5. In **Data Source**, select the database connection that contains the target tables.

6. Click **Create Historian**.

Create Historian

Create Historian

1. Select Type

2. Configure Historian

Configure DB Table Historian

MAIN

Name *

DA Table Historian

Description

Enabled

☒ Set whether this historian is enabled or disabled within the Gateway.

Data source

mySQL

Back

Create Historian

Retrieving Records

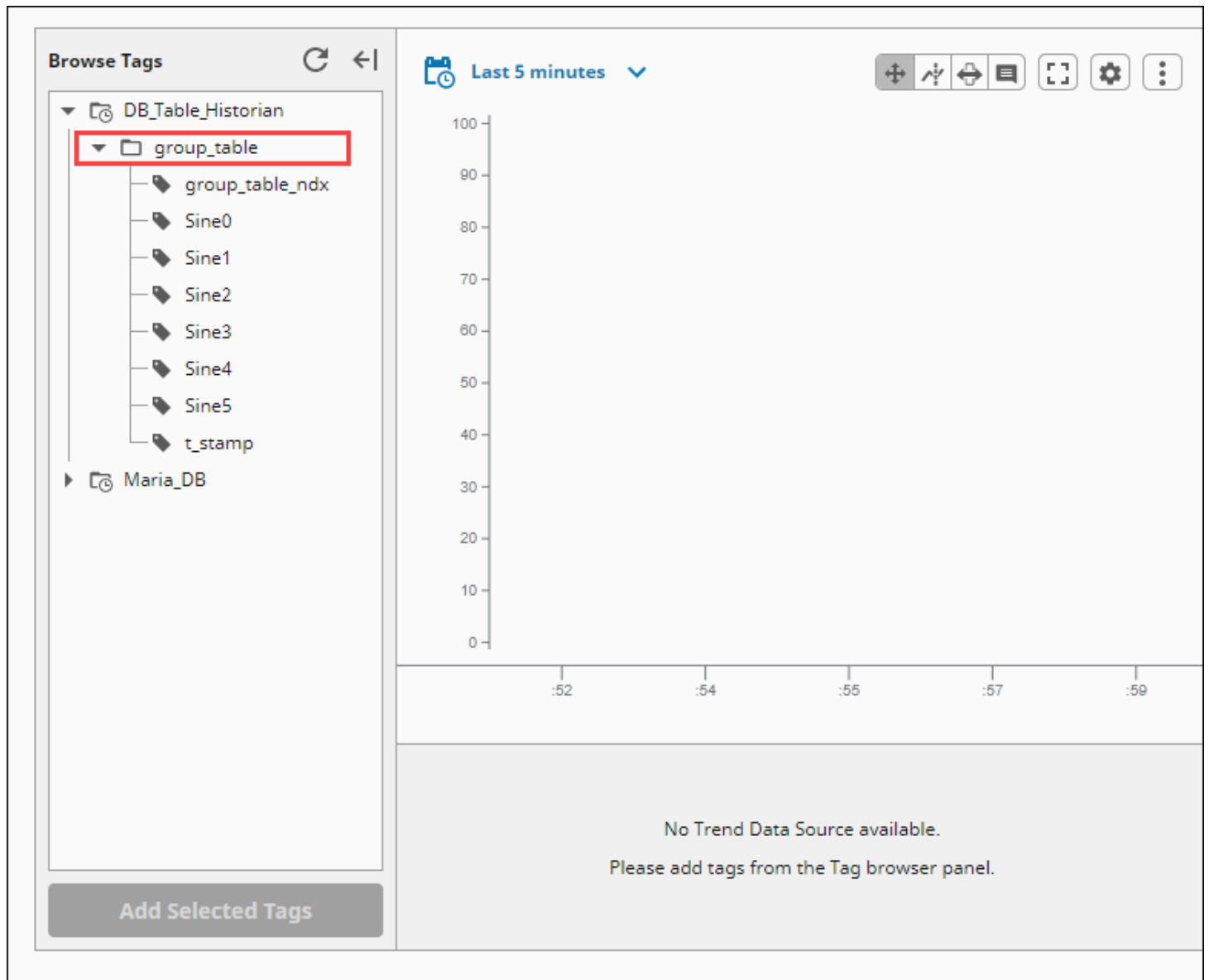
Once configured, the DB Table Historian will expose tables from the selected data source to tag history query tools like the Vision Tag History binding or the Perspective Power Chart.

For example, a Transaction Group might generate a table like this in the database schema:

Schema	History
▼ group_table	
group_table_ndx	(INTEGER)
Sine0	(REAL)
Sine1	(REAL)
Sine2	(REAL)
Sine3	(REAL)
Sine4	(INTEGER)
Sine5	(INTEGER)
t_stamp	(TEXT)

In this example, the group_table contains several columns (e.g., Sine0–Sine5 and t_stamp) that represent recorded tag values and timestamps. These columns can be queried just like standard historical tag data.

Then, we can use a Perspective Power Chart to detect the table and expose the columns as tags.



Here, the `group_table` is detected under the DB Table Historian Provider. Each column appears as a selectable tag that can be added to the chart for historical analysis, just like standard historian data sources.

Path Components

Historical tag paths for the DB Table Historian are made up of several components. These paths allow Ignition to interpret database tables as if they were standard historical tag records.

Each component in the path defines how the system locates and interprets the relevant data within the target table.

Component	Description
histprov	Name of the DB Table Historian Provider.
table	Table name in the data source.
column	Column in the table to query.
timestamp	Column to use for timestamps. Defaults to t_stamp if not specified.
keycolumn	Column for basic filtering. Only one keycolumn per path. [Optional]
keyvalue	Value to filter by in keycolumn. Must be an integer. [Optional]

i NOTE

For more advanced filtering, use a [Named Query](#) instead of a tag history path.

Here's an example path:

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

Example: Displaying Custom Historian Data in a Power Table

You can use the DB Table Historian Provider to expose historical data stored in a standard SQL table. This example shows how to configure a Vision Power Table to show that data using a Tag History binding.

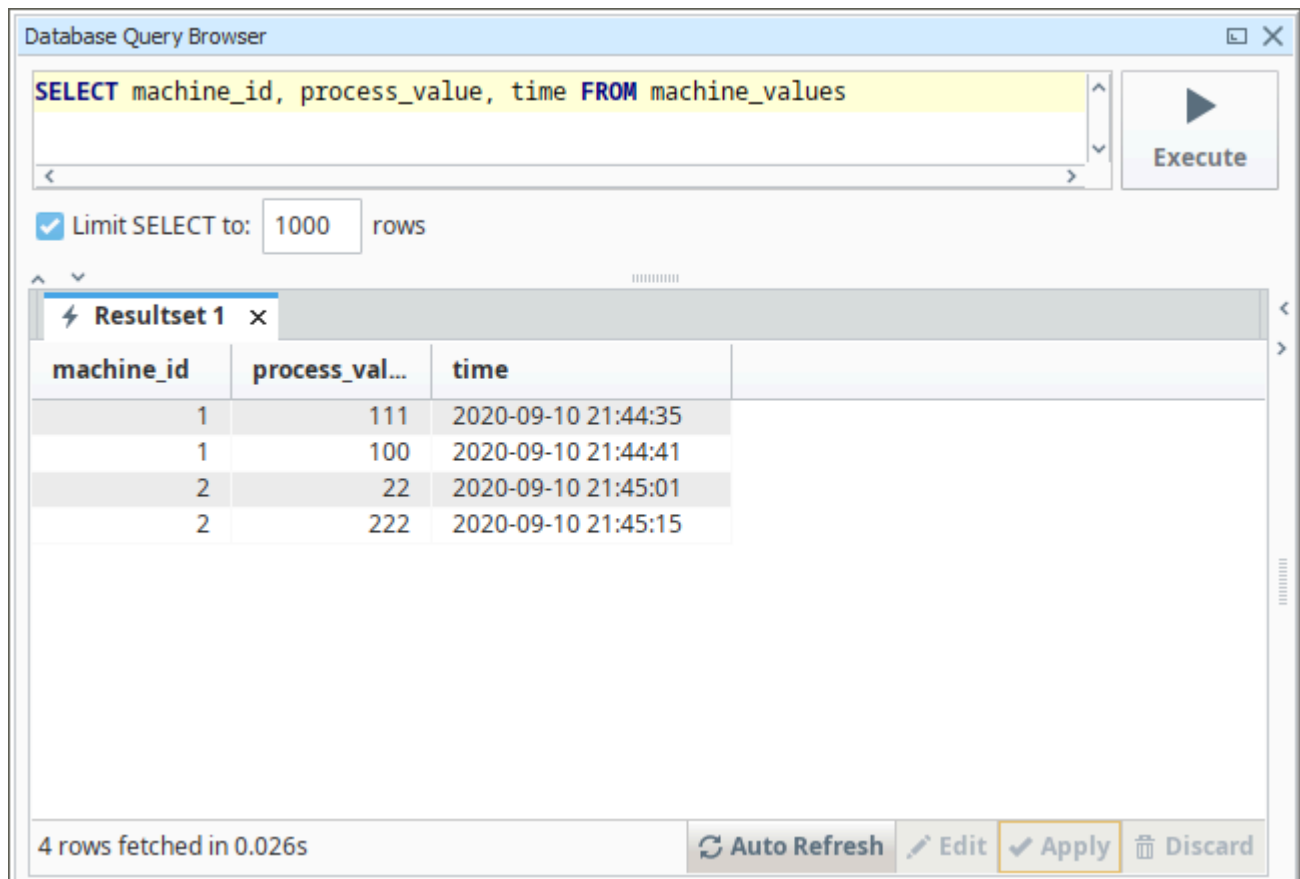
1. Create a table in your database to hold historical values:

```
CREATE TABLE machine_values (  
  machine_id INT,  
  process_value INT,  
  time DATETIME  
);
```

Insert sample data:

```
INSERT INTO machine_values (machine_id, process_value, time) VALUES
(1, 111, '2020-09-10 21:44:35'),
(1, 100, '2020-09-10 21:44:41'),
(2, 22, '2020-09-10 21:45:01'),
(2, 222, '2020-09-10 21:45:15');
```

You can view the table and its contents using your database's query browser. The following image shows the result in a typical SQLite browser interface, displaying the `machine_values` table and its data:



2. In the Designer, add a Power Table component to your window.
3. Navigate to the `data` property and click the Binding icon.
4. Choose **Tag History** as the binding type.
5. Use the **Browse Tags** button to select your historical tag. If the provider is correctly configured, you'll see columns from your database listed as tags.
6. Use tag history path syntax to refine the query. For example, the following paths tell the historian where to find values, which column represents time, and what rows to include.
 - Basic path for a column:

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

- Specify a timestamp column:

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

- Filter to rows where `machine_id = 1`:

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

7. After selecting your tag and setting the options, click **OK** to apply the binding.

The Power Table will now display the historical `process_value` data for `machine_id = 1`.

Property Binding: Root Container.Power Table

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

Available Historical Tags

- DB Table Historian
 - group_table
 - machine_values
 - machine_id
 - process_value**
 - time
 - internalDB

Selected Historical Tags

Tag Path	Column Name
histprov:DB Table Historian:/table:machine_values:/column:process_value	process_value

Indirection

Ref. #	Property Path
--------	---------------

☐ Use fully-qualified paths

Date Range **Most Recent**

Realtime 10

OK Cancel

Simulator Historian



INDUCTIVE
UNIVERSITY

Historian Simulator

[Watch the video](#)

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

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

Configuring a Simulator Historian Provider

1. On the Gateway Webpage, navigate to the **Services** section.
2. Under the **Historians** heading in the sidebar, click on **Historians**.
3. Click **Create Historian**.
4. From the listing of available types, select **Simulator Historian**, and click **Next**.

Create Historian

Create Historian

1. Select Type

2. Configure Historian

☐ Internal Historian - QuestDB

A built-in local historian optimized for high-throughput, demanding workloads. Recommended choice for most use cases that require local timeseries data storage.

☐ Internal Historian - SQLite

A built-in local historian for lightweight applications.

☐ OPC HDA Historian

A historian that uses OPC-HDA to retrieve values from a third party historian.

☐ Remote Historian

Stores historical data to a remote historian using the Gateway Network.

☒ Simulator Historian

Provides simulation data for testing without an external data source.

☐ SQL Historian

A historian that leverages a SQL database for the storage and retrieval of historical data.

Cancel

Next

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

Create Historian

Create Historian

1. Select Type

2. Configure Historian

Configure Simulator Historian

MAIN

*Required

Name *

Simulator Historian

The name of this tag historian.

Description

Enabled

☒ Set whether this device connection is enabled or disabled within the Gateway.

Back

Create Historian

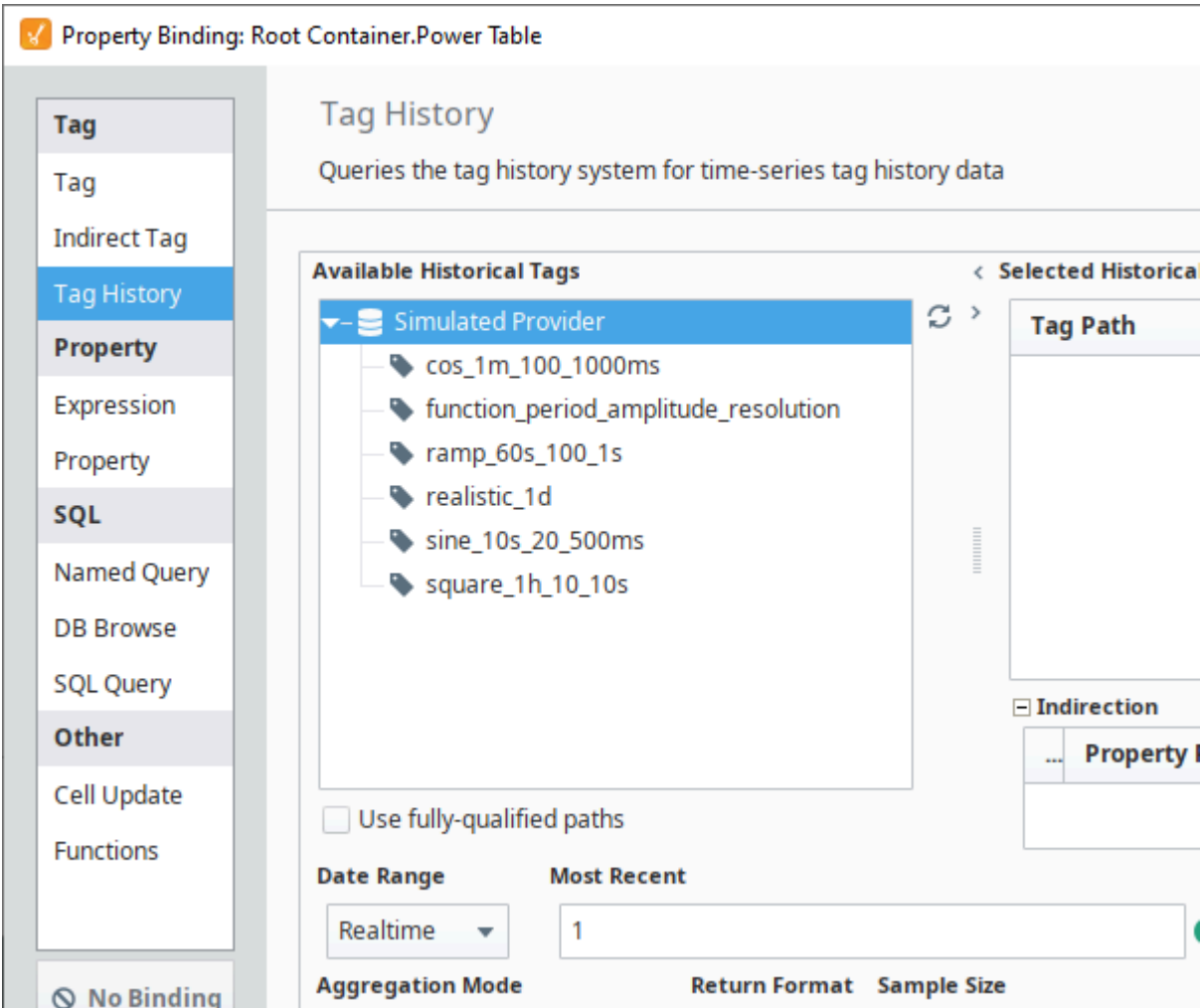
You'll be redirected back to the History Provider listings. You're now ready to use the simulator.

Retrieving Simulated Data

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

Simulated Provider

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



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

the tags utilizes the simulator's pathing syntax.

Simulator Path Syntax

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

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

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

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

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

```
function_periodTime_amplitude_resolutionTime
```

Each parameter is described below:

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

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

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

Editing the Paths

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

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

Property Binding: Root Container.Power Table

Tag

Tag

Indirect Tag

Tag History

Property

Expression

Property

SQL

Named Query

DB Browse

SQL Query

Other

Cell Update

Functions

Tag History

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

Available Historical Tags

Simulated Provider

cos_1m_100_1000ms

function_period_amplitude_resolution

ramp_60s_100_1s

realistic_1d

sine_10s_20_500ms

square_1h_10_10s

Use fully-qualified paths

Date Range

Most Recent

Selected Historical Tags

Tag Path

Co...

histprov:Simulated Provider:/tag:realistic_30m

...

Indirection

Ref. #

Property Path

Function Examples

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

```
tag:sine_10s_20_500ms
```

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

```
tag:square_1d_10_5m
```

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

```
tag:realistic_30m
```

Core Historian



INDUCTIVE
UNIVERSITY

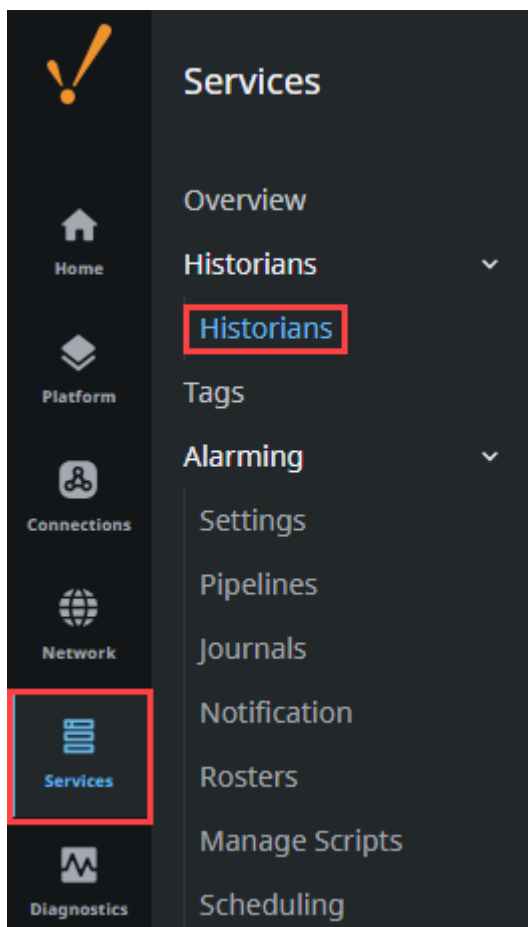
Core Historian

[Watch the video](#)

The Core Historian, powered by QuestDB, supports partitioning, deduplication, archiving, and native aggregation. This page demonstrates how to create a Core Historian on the Gateway, and describes the available settings and configuration options for existing historians.

Configuring a Core Historian

1. Navigate to the **Services > Historians > Historians** section of the Gateway.



2. On the **Historians** page, click **Create Historian +**.
3. From the Create Historian form, select **Core Historian**, then click **Next**.

Create Historian

Create Historian

1. Select Type

2. Configure Historian

☒ **Core Historian**
A built-in local historian optimized for high-throughput, demanding workloads. Recommended choice for most use cases that require local timeseries data storage.

☐ **CSV Historian**
A historian implementation that provides historical data from a CSV file.

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

☐ **Historian Splitter**
Writes historical data simultaneously to two historians while managing query routing between them.

☐ **Internal Historian (Legacy)**
A built-in local historian for lightweight applications.

☐ **OPC HDA Historian**
A historian that uses OPC-HDA to retrieve values from a third party historian.

Cancel

Next

4. Give your Historian an appropriate name, and configure the settings to align with your needs. In this example, we will name our Historian **Core Historian**, and leave the settings as the default. Click **Create Historian** when done.

Create Historian

Create Historian

1. Select Type

2. Configure Historian

Configure Core Historian

MAIN

Name *

Core Historian

Description

Enabled

☒ Set whether this historian is enabled or disabled within the Gateway.

Partition Interval *

Months

Data Deduplication

☐ When enabled, incoming data with matching keys will update existing entries rather than creating new ones.

MAINTENANCE

Back

Create Historian

Your Core Historian will now be listed and ready to store data.

Historians

Filter historians...

View 20

Create Historian +

NAME	TYPE	ENABLED	DESCRIPTION	STATUS	
Core Historian	Core Historian	true		✓ RUNNING	⋮
Simulator	Simulator Historian	true		✓ RUNNING	⋮

Showing 2 of 2 items

i MEMORY ALLOCATION

The default memory allocation for the Core Historian is 10% of the total system memory when enabled. Use the Java parameters described in the [Core Historian](#) section of the Gateway and Gateway Network Parameters page to adjust Core Historian memory allocation.

Settings

Main

Property	Description
Name	Specifies the unique name used to identify the historian provider.
Description	Provides an optional user-defined description for the provider.
Enabled	Set whether this historian is enabled or disabled within the Gateway.
Partition Interval	Granularity of table partitioning (Week, Month, Year).
Data Deduplication	When enabled, incoming data with matching keys will update existing entries rather than creating new ones.

Maintenance

Property	Description
Mode	Determines how older data is archived or permanently deleted. Options include: • None: History is stored indefinitely and stays in the same location. It isn't archived or deleted. If this mode is selected, no other properties need to be set under the Maintenance section. • Prune: Data will be removed by deleting partitions based on the configured age threshold values. This is the default value. • Archive: Data partitions are removed, and then retained in a new target location based on the configured age threshold values.
Archive Folder	The path to a folder that will serve as the target location for the archived partitions. This setting is only available when the Mode property is set to Archive.
Maintenance Age	Specifies the age threshold for data maintenance.
Maintenance Age Units	Specifies the time unit used for the data maintenance age threshold. Must be a whole number multiple of the Partition Interval, e.g., if the Partition Interval is DAY, the Maintenance Age must be a whole number of days (24 HOURS, 2 DAYS, 3 MONTHS, etc.)

Metadata Storage

When enabled on a tag, metadata is stored automatically in system-managed tables. This includes properties like tag path, data type, units, and scaling. Metadata is preserved even if the tag is removed.

Metadata must be explicitly enabled in the Tag Editor by setting **Include Metadata** to true. This setting is found in the **History** section when editing a tag in the Designer.

Metadata is required for features like annotations, aggregates, and Historian scripting functions.

Customizing Server Settings

New in 8.3.4

Starting in 8.3.4, `server.conf` and `log.conf` files will be automatically created with a Core Historian connection. These files allow for certain server settings to be customized, such as the Cairo commit mode and log routing. Note that changes to the `server.conf` file do not change Core Historian settings, but the embedded QuestDB configuration file instead.

Once a connection has been created, these files can be found in the

`%IgnitionInstallationDirectory%/data/var/com.inductiveautomation.historian/corehistorian/conf/` directory.

The `log.conf` file provides file and stdout writer logging properties. The `server.conf` file contains a filtered set of properties for the following settings:

- RAM
- Cairo
- Materialized View
- WAL
- HTTP
- PG Wire
- Metrics
- Logging

Some of these settings are restricted and set to `false` by default. The default `cairo.commit.mode` property is set to `sync`. Refer to the [QuestDB Configuration](#) page for more information on the `server.conf` file.

LEGACY SYSTEM PROPERTIES

Support for Ignition's legacy system properties that relate to historian configuration, such as `historian.questdb.httpServerEnabled` and `historian.questdb.pgwireServerEnabled` have been maintained for backward compatibility. If used, these properties will take precedence over any conflicting values set in the `server.conf` file.

SQL Bridge (Transaction Groups)

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

Transaction Group Overview

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

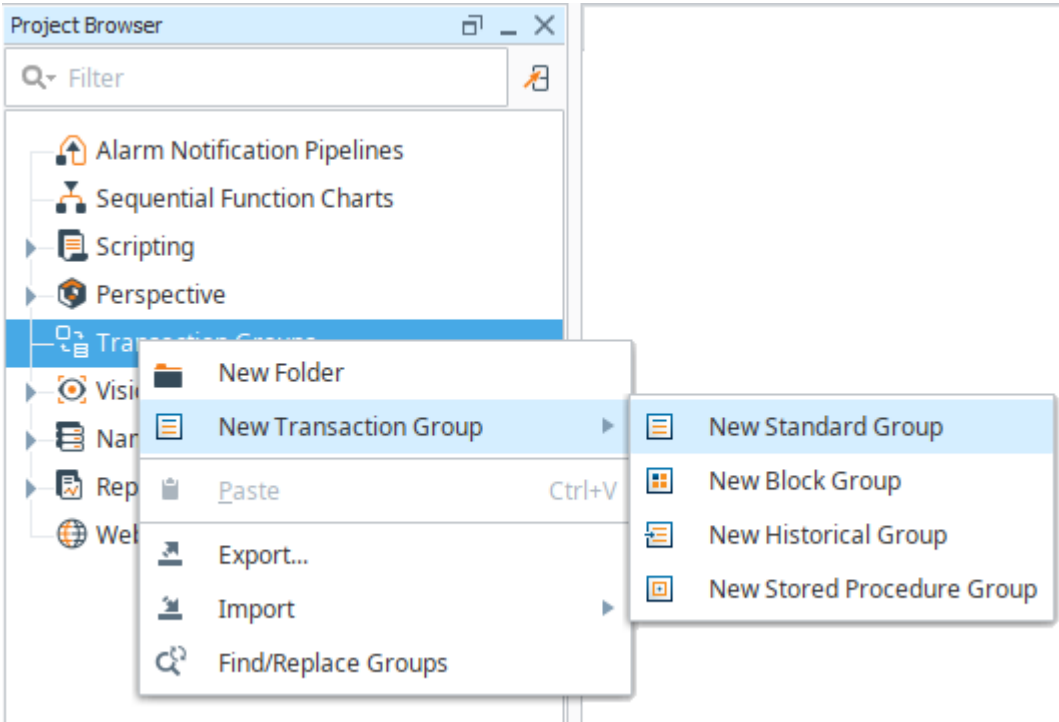
Transaction Groups are configured in the Ignition Designer. The group will then execute at a specific interval of time, or on a user-defined schedule. A **trigger** can be used to prevent planned executions when specified conditions are not met. On execution, the simplest groups (Historical groups) will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

Transaction Group Applications

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

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

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



Historical Data Logging

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

new historical

Running

Enabled

Disabled

Pause

Basic OPC/Group Items (6)					
Item Name	Source Va...	Latched ...	Target Name	Data Type	Properti...
Sim_Generic/Ramp/Ramp0	344.427	344.427	Ramp0	Float8	
Sim_Generic/Ramp/Ramp1	58.320	58.320	Ramp1	Float8	
Sim_Generic/Ramp/Ramp2	119.427	119.427	Ramp2	Float8	
Sim_Generic/Ramp/Ramp3	247.480	247.480	Ramp3	Float8	
Sim_Generic/Ramp/Ramp5	465.267	465.267	Ramp5	Float8	
Sim_Generic/Ramp/Ramp4	158.320	158.320	Ramp4	Float8	

Action

Trigger

Options

Execution Scheduling:

Timer

Schedule

1 second(s)

Data source:
MySQL

Table name:
ramp_historical

☒ Automatically create table

Database Query Browser

SELECT * FROM ramp_historical

Limit SELECT to: 1000 rows

Execute

Resultset 1

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

268 rows fetched in 0.013s

<Default>

Schema

History

SELECT * FROM Tank_Histo

SELECT * FROM ramp_histo

Auto Refresh

Edit

Apply

Discard

Recipe Management and Synchronization

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

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

Line 1 Recipe

Running

Enabled

Disabled

Pause

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

Block Data Transfers

The [Block Group](#) supports the transfer of entire arrays of data to and from the database.

T4 ACC

Execution Disabled

Enabled

Disabled

Pause

Item View

Block View

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

Stored Procedures

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

SP All Params

Errored

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

Understanding Transaction Groups



INDUCTIVE
UNIVERSITY

Transaction Group Introduction

[Watch the video](#)

Transaction Group Workspace

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

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

Tank Data

Running

Title Bar

▶ Enabled ⓧ Disabled || Pause

Basic OPC/Group Items (3)

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

Run-Always Expression Items (ignore trig

Item Tables

Item Name	Source Val...	Name	Data Type	Properties
-----------	---------------	------	-----------	------------

Triggered Expression Items (0)

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

⚡ Action ▶ Trigger ⚙ Options

Execution Scheduling

☒ Timer ☐ Schedule

1 second(s)

Up

0

Da

<|

Table name:

group_table

☒ Automatically create table

☐ Use custom index column: group_table_nc

☒ Store timestamp to: t_stamp

☐ Store quality code to: quality_code

☐ Delete records older than:

⚡ Status ⚙ Events

Status/Events Tabs

Running

Last execution: Sat Mar 12 00:16:21 UTC 2022

Total executions: 68

Last trigger: Sat Mar 12 00:16:21 UTC 2022

Failed executions: 0

Next execution: Sat Mar 12 00:16:22 UTC 2022

OPC/Tag writes: 0

Last duration: 0.0 second(s)

DB writes: 272

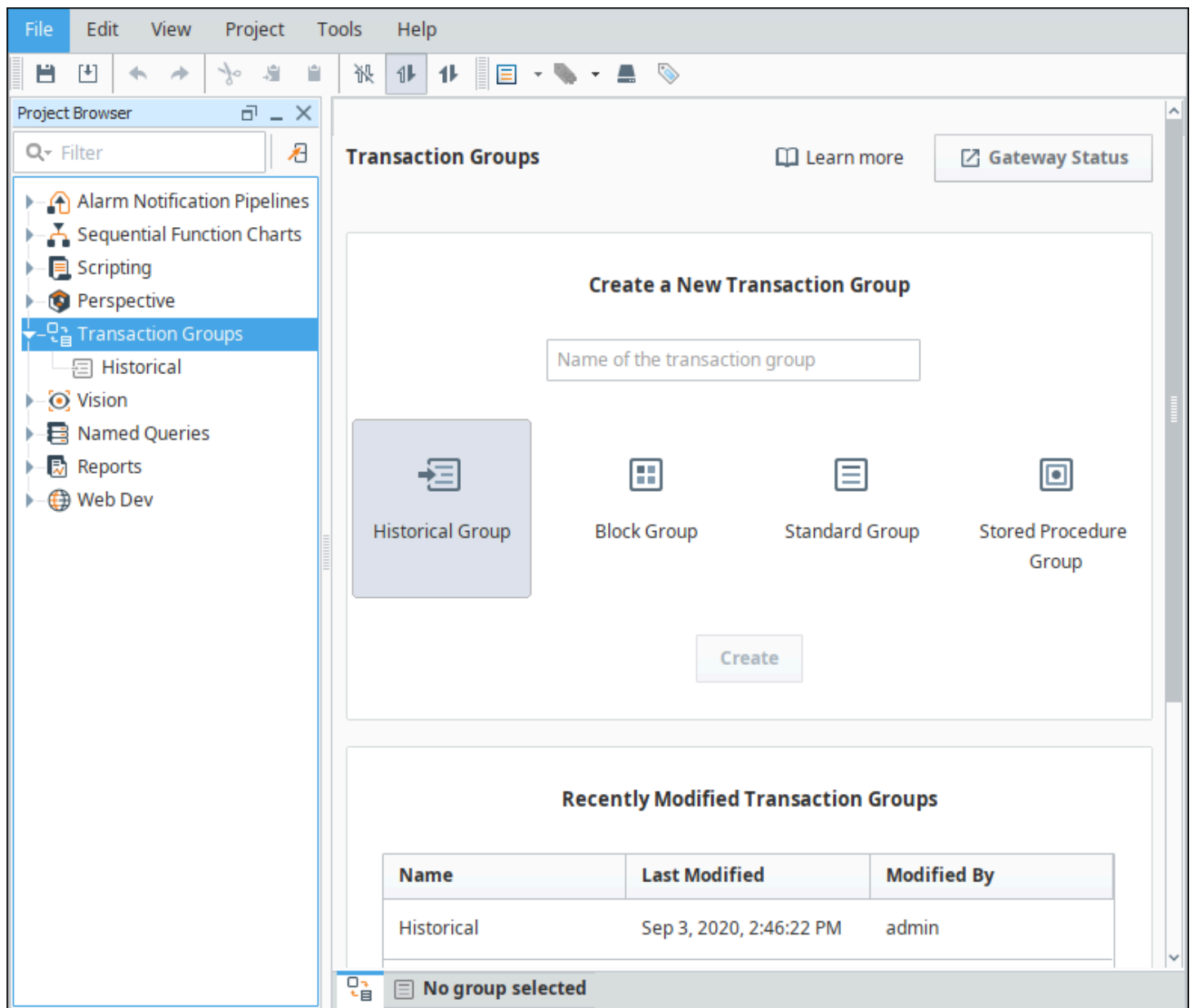
Avg duration: 0.00030 second(s)

OPC/Tag write failures: 0

Tank Data

Welcome Page

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



Project Browser

Status icons will also display next to the Transaction Group name in the Project Browser. These icons will update as you edit and save your Transaction Group to indicate running, modified, disabled, and paused statuses. Hover over an icon to display its tooltip for more context.

Items

Each Item (tag) in the Transaction Group consists of several properties, which are sorted by column. The key columns are the Source Values, Latched Values, and Target Name. The remaining columns are more straight-forward in their purpose. The Data Type column lists the data type of the group item while the Properties column displays any additional configurations made to the item, such as alarming or a specified function.

Source and Latched Value

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

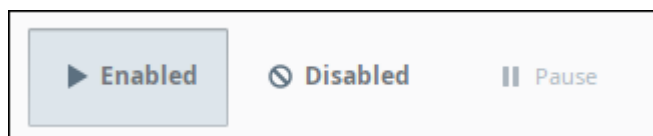
The Latched value will be the value that was written at execution. This can be the value that gets written to the database on execution in OPC to DB mode, or it can be the value that gets written to the tag in DB to OPC mode. The value will only change on execution of the group.

Target Name

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

Enabling Group Execution

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



NOTE

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

Editing Group Settings

Group settings may be modified at any time, regardless of whether or not the group is executing. Modifications will be applied when the project is saved, and the group will be started or stopped as

required. Some changes such as modifying items may cause features like live values to appear to be incorrect. It is therefore important to notice the modified icon that appears next to the group, and to save often. If you would prefer to stop the group before making edits you can simply pause the group. Execution will begin again after the project is saved.

Action Items

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.

Action Trigger Options

Execution Scheduling:

☒ Timer ☐ Schedule

1 second(s)

Update mode:

Bi-directional OPC wins

Data source:

<Default>

Table name:

sample

☒ Automatically create table

☐ Use custom index column: sample_ndx

☒ Store timestamp to: t_stamp

☐ Store quality code to: quality_code

☐ Delete records older than: 1 day(s)

Table action

☒ insert new row

☐ update/select

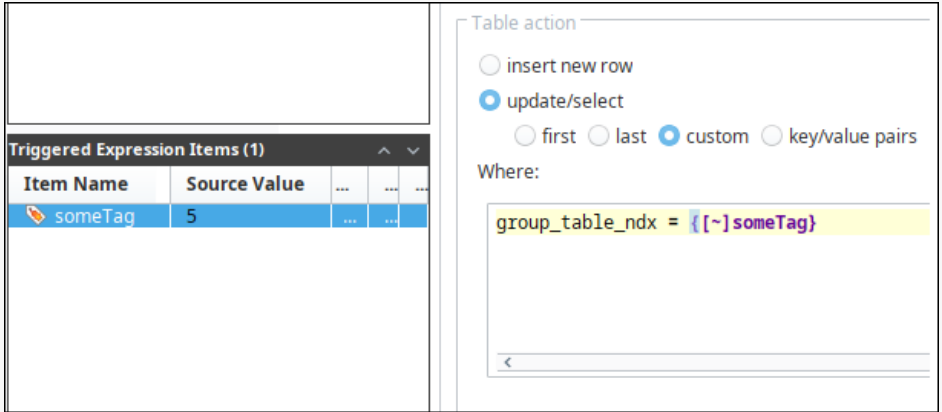
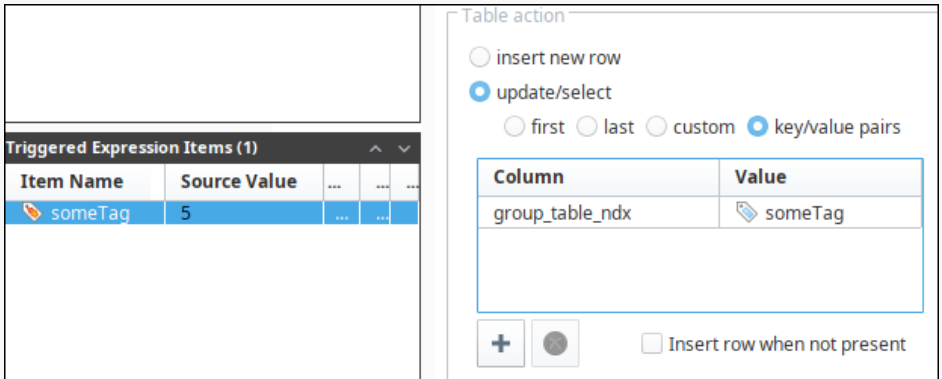
☒ first ☐ last ☐ custom ☐ key/value pairs

Where:

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	Despite the name, determines how often the group is <i>evaluated</i>. For a number of reasons, the group may not execute during the evaluation. The most common reason is the trigger, but see Execution Cycle below for more possible reasons why evaluation will exit. • Timer - specifies the OPC tag subscription rate for the OPC tags. It can run at millisecond, second, minute, hour, or day rates. • Schedule - is a specified start time on the update Rate. Set a list of time (or time ranges) that the group should evaluate at. If the pattern specified includes a time range, a rate must be provided.
Update mode	For groups that support it, sets the default for how items are compared to their targets. Options are: • 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.
Data source	The database connection name the group should use. Can be Default, which will use the default connection for the project.
Table name	Name of the table in the database that the group should interact with (reading or writing, depending on the Update mode and individual item Mode settings). The tables listed in this dropdown are determined by the Data source property. This setting allows you to type arbitrary names into it. If you type the name of a database table that doesn't exist, and the Automatically create table setting is enabled, then the group will attempt to create the database table on start.
Automatically create table	If enabled, the Transaction Group will attempt to create a database table once the group starts running, assuming one doesn't already exist as

Setting	Description
	determined by the Table name setting. If the table already exists, then nothing happens.
Use custom index	If left disabled, the group will attempt to add an index column to the database table when the group starts executing. If enabled, the group will use the column selected in the adjacent dropdown, or create a new column if you type in a column name that doesn't exist on the table (requires the Automatically create table setting to be enabled).
Store timestamp	If enabled, will attempt to store a timestamp value to the column specified in the adjacent dropdown. If you type in a column name that doesn't exist on the table, the group will attempt to create the column on start, assuming the Automatically create table setting.
Store quality 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). While enabled, a task will be created to check the database tables for older records. The rate this task occurs is determined by frequency configured on this property. If the configured frequency is: • Less than an hour: Task executes every minute • Between one and twelve hours: Task executes every 15 minutes • Over twelve hours: Task executes every hour Transaction Groups using this property will use timestamps instead of the row index the record was inserted at when pruning.
Table Action	Defines which row will be targeted by the group. • insert new row • update/select - allows you to target specific rows in the database table. Options are: ◦ first row - the group always executes against the first row. ◦ last row - the group always executes against the last row. ◦ custom - allows you to write a custom WHERE clause to determine which row should be targeted. Uses the Where text

Setting	Description
	area. The custom clause can use references to values of items in the group.  ◦ key/value pairs - Provides dropdowns for both a column and an item in the group, allowing the group to target a single row in a table based on the item's value. For example, if a value of 5 is used in conjunction with the "group_table_ndx" column in the database table, rows where "group_table_ndx" has a value of 5 are targeted when the group executes.  Additional conditions can be added or removed with the Add or Delete buttons at the bottom of the Table Action settings.

Group Update Rate



INDUCTIVE
UNIVERSITY

Group Update Rate

[Watch the video](#)

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

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

Schedule

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

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

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

When using ranges, the execution times will be aligned to the start time. For example, if you specify a schedule of "9am - 5pm" with a rate of "30 minutes", the group will evaluate at 9, 9:30, 10, etc., regardless of when it was started. This is a useful difference compared to the Timer mode, which runs based on when the group was started. For example, if you want a group that runs every hour, on the hour, you could specify a 1 hour rate with a range of "0-23."

Execution Cycle

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

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

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

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

CAUTION

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

Trigger Settings

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

<

⚡ Action

▶ Trigger

⚙ Options

>

☒ Only evaluate when values have changed.

Tags to watch for change:

All tags

Select tags

☐ Execute this group on a trigger

Trigger on item:

☐ Only execute once while trigger is active

☐ Reset trigger after execution

☐ Prevent trigger caused by group start

Trigger conditions:

☐ is != 0 (or true)

☒ is = 0 (or false)

☐ is active:

>

0

☐ non-active:

<=

1

☐ Active on value change

☐ Write handshake on success

Set:

To value:

0

☐ Write handshake on failure

Set:

To value:

0

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

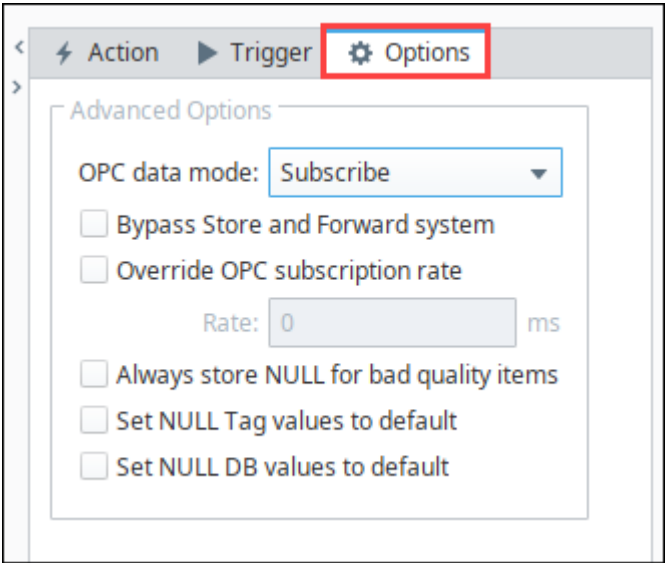
Setting	Description
Only evaluate when values have changed	Enabling this setting will cause the group to evaluate only if values or Tag qualities have changed. If the values have not changed, it will exit the evaluation. You have the option to monitor all Run-Always items in the group, or only specific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

Setting	Description
	Active on value change, which will cause the group to execute if the trigger changes value at all.
Write handshake on success	Set the item and the value you want to see when the group executes successfully.
Write handshake on failure	Set the item and the value you want to see when an error prevents the group execution.

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	Modifies how the group receives data from OPC. Options are Subscribe or Read. • Subscribe - Data points are registered with the OPC server, and data is received by the group on-change. 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. Note: 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. • Read - 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. Note 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. Additionally, while in Read mode, the tags selected for the Tags to watch for change setting will be subscribed to as well.
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

Setting	Description
	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. 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. The following types will have the following corresponding values: • 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)

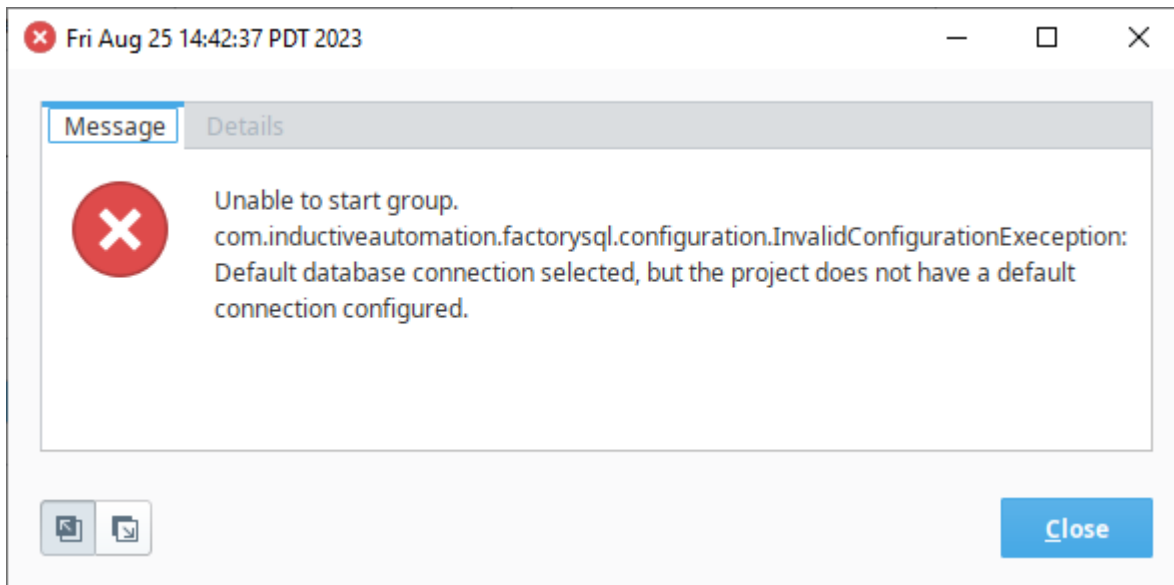
Troubleshooting

It may be helpful when troubleshooting or testing Transaction Groups to increase the default thread pool count. Refer to the [Gateway Configuration File Reference - Thread Pool Counts](#) for more information.

Events Tab

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

Status	Events
Message	
	Group state changed to 'warned'
	Group state changed to 'running'
	Unable to start group.
	Group state changed to 'fatally_errored'



Creating a Transaction Group



INDUCTIVE
UNIVERSITY

Basic Historical Group

[Watch the video](#)



INDUCTIVE
UNIVERSITY

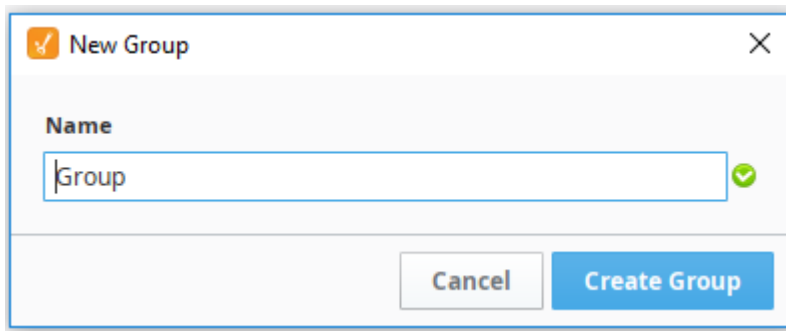
Realtime Group

[Watch the video](#)

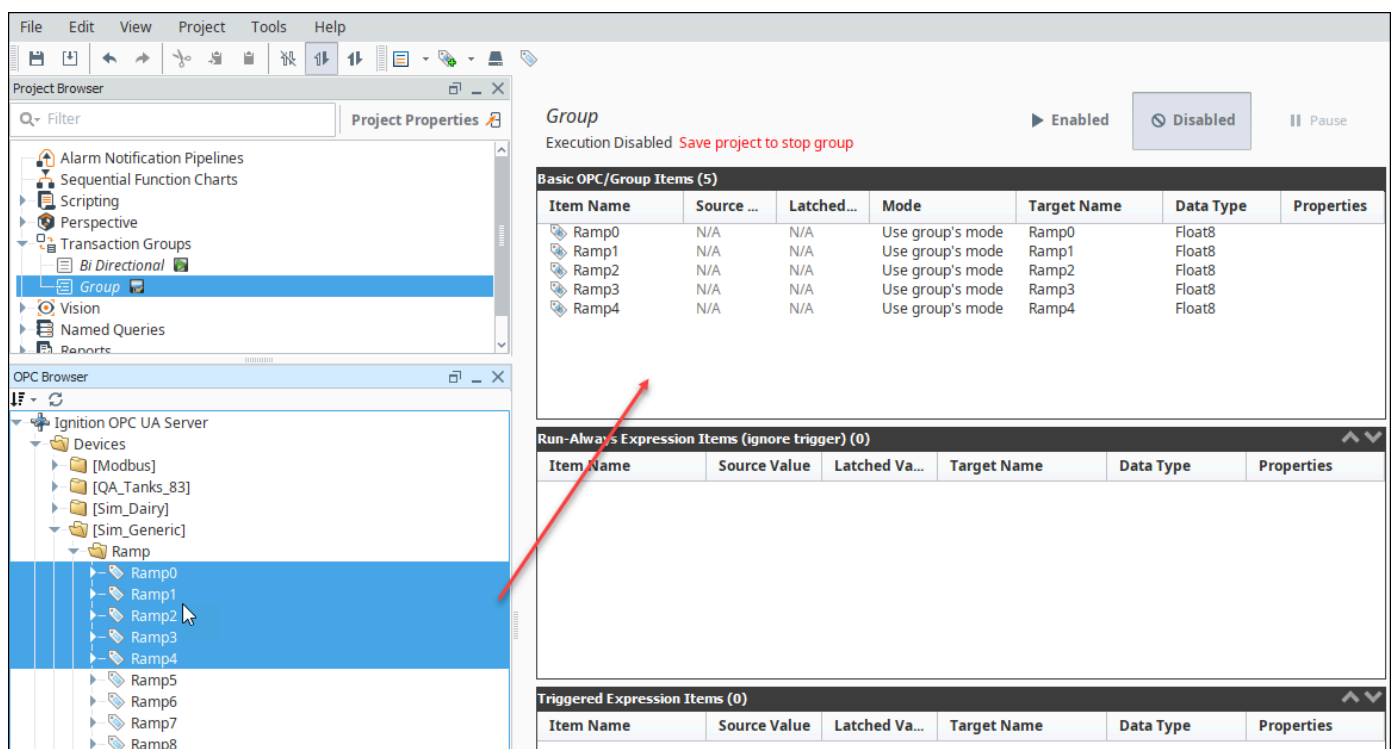
This example demonstrates how to configure a transaction group, specifically a Historical Group. However, the process of creating any transaction group type is very similar, especially so in the case of a standard group. The [Transaction Group Examples](#) section contains more examples.

1. Click on the **Transaction Groups** in the Project Browser to switch the Designer's workspace to the Transaction Group workspace.

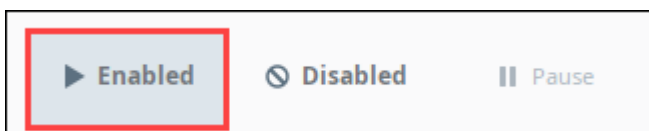
- In the Project Browser, right click on **Transaction Groups > New Transaction Group** to make a New Historical Group. Name the group 'Group.'



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



- Save your project.
- Click the **Enabled** button above the item ables to enable logging.



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

⚡ Action ▶ Trigger ⚙ Options

Execution Scheduling:

☒ Timer ☐ Schedule

1 second(s)

Data source:

<Default>

Table name:

New_Test_Table

☒ Automatically create table

☐ Use custom index column: group_table_ndx

☒ Store timestamp to: t_stamp

☐ Store quality code to: quality_code

☐ Delete records older than:

1 day(s)

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

Database Query Browser

SELECT * FROM group_table_new WHERE Ramp0>0

Execute

☒ 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

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.

Standard Group

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

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

Group Settings

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

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

Table Action

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

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

Typical Uses

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

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

Historical Group

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

Group Settings

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

Typical Uses

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

Block Group



INDUCTIVE
UNIVERSITY

Block Group Type

[Watch the video](#)

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

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 (integer, float, string), create a copy of the item that stores item path, and then use the insert changed rows option to create your own vertically storing historical tables. Create additional copies of the block item that refer to quality and timestamp in order to get further information about the data point.

Table Format

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

table_ndx	tag1	tag2	tag3
1	10	20	30

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

Blocky

Running

▶ Enabled

⛔ Disabled

⏸ Pause

Item View

Block View

Block Items (1)

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

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

table_ndx	Tags
1	10
2	20
3	30

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

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

Row ID and Block ID

Using the same tag examples 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 (rows 1-3) are part of the same execution, and rows with a block_id value of 2 (rows 4-6) are a separate execution.

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

Group Settings

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

The unique settings are:

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

Table Action

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

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

Stored Procedure Group



INDUCTIVE
UNIVERSITY

Stored Procedure Group Type

[Watch the video](#)

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

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

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

NOTE

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).

SP All Params

Errored

▶ Enabled

⏻ Disabled

⏸ Pause

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

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

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

Group Settings

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

- **Store timestamp to:** Specifies whether or not to store a timestamp with the record, and the target column. The timestamp will be generated by the group during execution. For groups that update a row, the timestamp will only be written if any of the values in the group are also written.
- **Store quality code to:** If selected, stores an aggregate quality for the group to the specified column. The aggregate quality is the combined quality of all of the items that write to the table. For more information about quality values, see [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 or PostgreSQL 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**.

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. Note: An OPC Item that is referenced in a Run-Always Expression Item will have a subscription created, regardless of Trigger or Tags to watch for change settings.
Expression Item	Much like an Expression tag, Expression Items are flexible in that their value can come from a number of different sources: specifically an expression or a database query. Expression Items have two subtypes: • Run-Always Expression Items are evaluated every time the group executes. Meaning, they'll run their associated expression or query every time the group executes. • Triggered Expression Items only evaluate when the group trigger is active.
Tag Reference Item	A reference to a tag in a Tag Provider. Allows a tag to be used in a group like any other item type, except that the tag is evaluated by its scan class instead of by the group. Tag Reference Items can reference the value on any tag in a Tag Provider, such as Query tags and Memory tags. For more information, see the Tag References vs. OPC Items section.

Tag References and OPC Items



INDUCTIVE
UNIVERSITY

Tag References vs. OPC Items

[Watch the video](#)

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

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

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

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

Expression Items



INDUCTIVE
UNIVERSITY

Expression Items

[Watch the video](#)

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

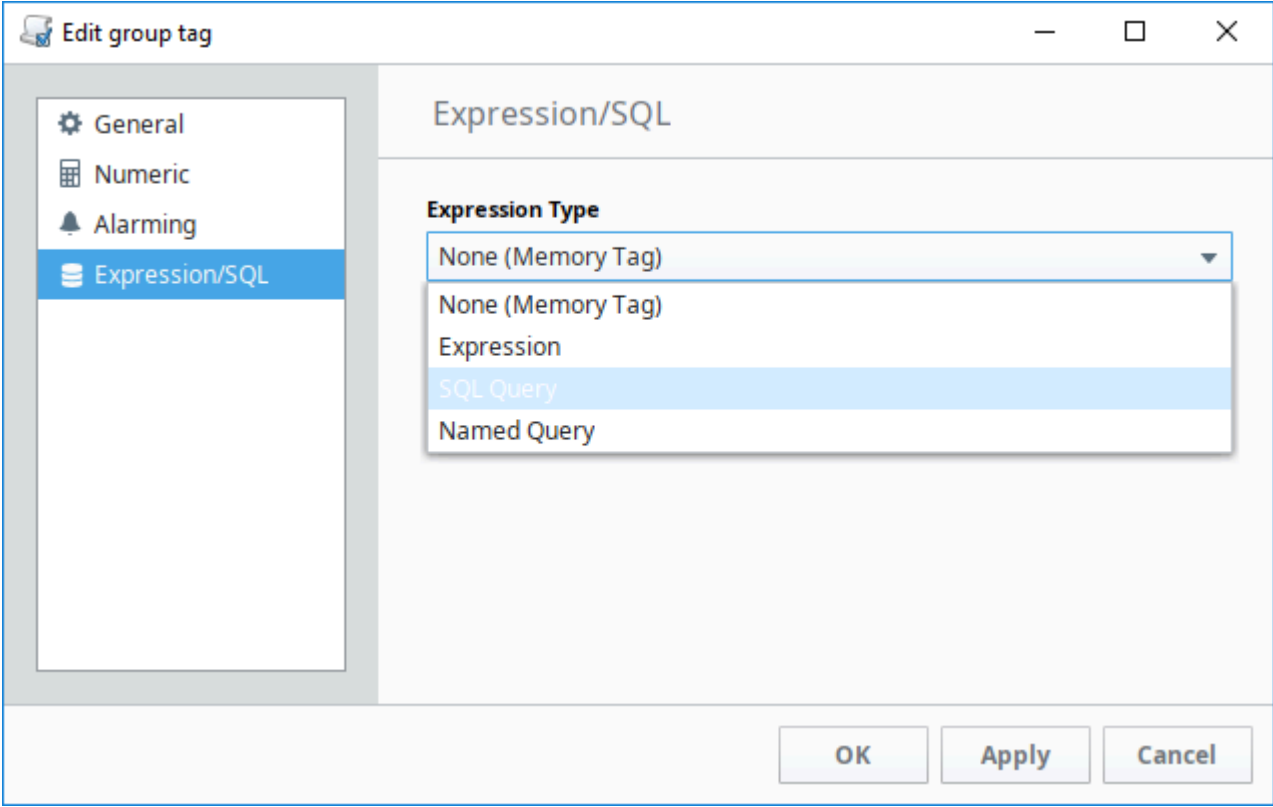
It is important to understand that an Expression Item only exists within its group, and can not be referenced by items in other Transaction Groups, Tags, and any components on a window.

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 Expression Language to determine the value on the item. The expression can reference other items in the group, as well as tgs.
SQL Query	Utilizes a SQL query to determine the item's value. Thus, a query can execute on the item and the results can be referenced by other items in the same group.
Named Query	Selecting this option will cause the value on the item to be determined by a Named Query in the same project as the Transaction Group.

Refer to the Property Table at the bottom of this page to see the [Expression Item Options](#).

Run Always vs. Triggered Items



INDUCTIVE
UNIVERSITY

**Run Always vs. Triggered
Items**

[Watch the video](#)

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.

Bi Directional

Unable to Start

▶ Enabled

⊘ Disabled

⏸ Pause

Basic OPC/Group Items (10)

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

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

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

Triggered Expression Items (1)

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

HOUR AND EVENT METER

Be aware that a Gateway or Transaction Group restart will reset the Source and Latched values of an Expression Item that uses [Hour or Event meter](#) modes. Any edits to the Expression Item, such as adjusting the execution schedule, will also reset these values.

Changing the Evaluation State

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

Edit group tag

General Properties

General

Numeric

Alarming

Expression/SQL

General Properties

Name
item 2

Value
0

Data Type
Integer

Value Mode

Property
Value

Mode

☒ Direct value

☐ Hour meter ☐ Event meter

☐ On Zero ☐ Retentive Units second(s)

☐ Reset on condition: > 0

Evaluation Options

☐ Run-always (ignore trigger)

Write target

Target Type
None, read-only item

Target Name

OK Apply Cancel

SQL Queries and Expressions



INDUCTIVE
UNIVERSITY

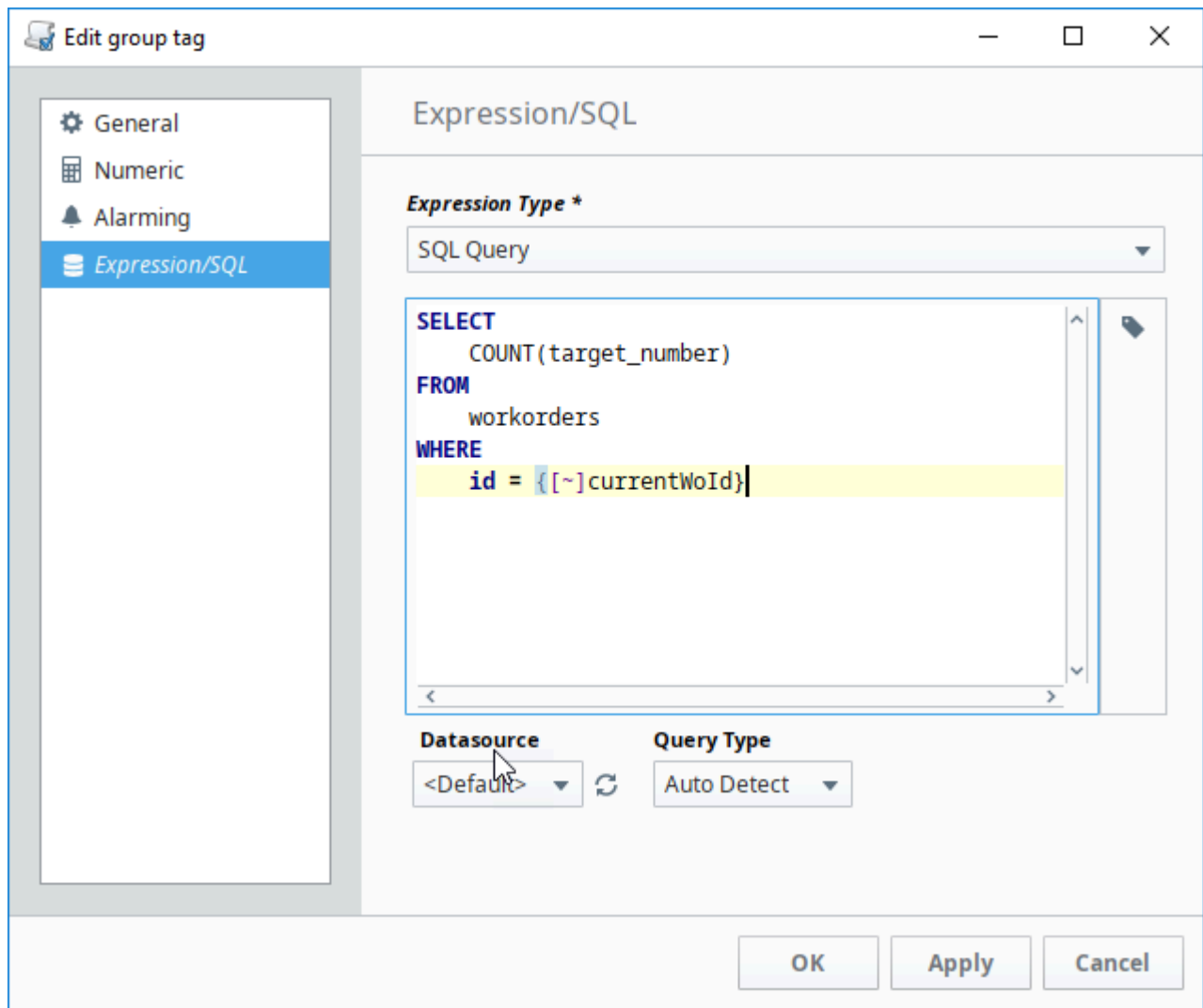
SQL Query Expression Items

[Watch the video](#)

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.

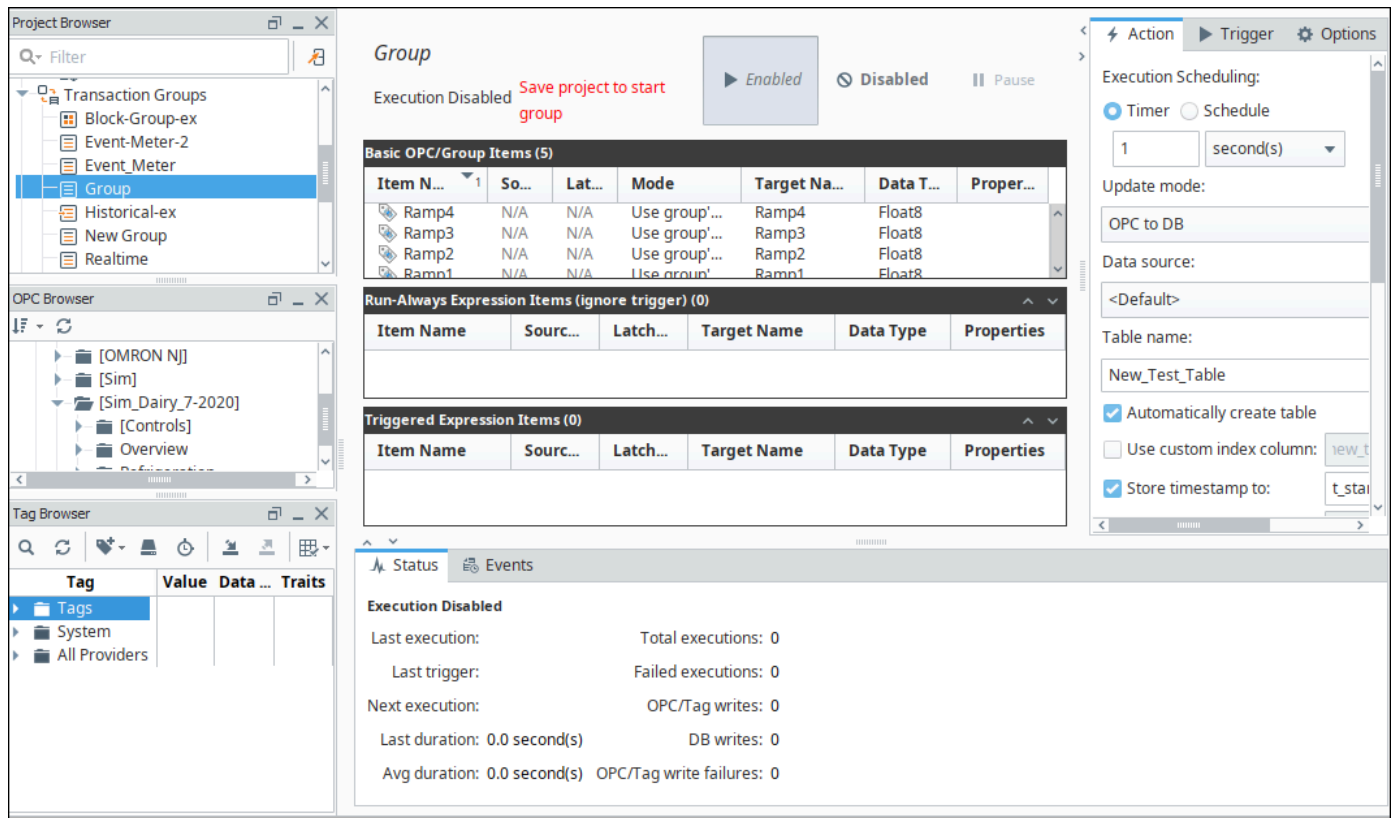
There are several Expression functions available that exist only for Transaction Groups. You can find them in the **Store and Forward** and **Variables** sections of the **f(x)** function list.



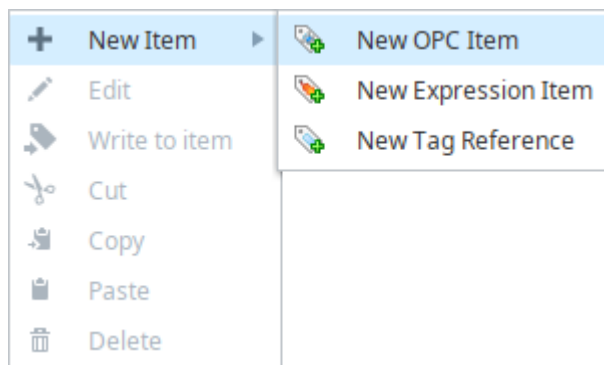
Creating a New Item

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:



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



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

Item Type Property Table

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

OPC Item Options

General

Property	Description
Name	The name of the OPC Item in the group. There cannot be duplicate names within a group.
Data Type	The data type used to read values from the PLC.

OPC Properties

Property	Description
OPC Server	The Selected OPC Server. This is a dropdown list showing all the OPC Servers added in the Ignition Gateway.
OPC Item Path	The OPC address assigned by the server. Dragging and dropping from the OPC Browser will automatically populate this field. You can escape curly braces <code>{ }</code> in the item path by using additional curly braces. For example: <code>{{device_name}}</code> would evaluate to <code>{<device_name param value>}</code>.
Source Data Type	Data type for the OPC Item.

Value Mode

Property	Description
Property	Which property of the OPC Item you want to use. • Value: Item value • Quality: Quality code from OPC Server (192 = GOOD_DATA) • Timestamp: The last time the item value changed. • Name: The SQLBridge Item Name property of this item. Note: UDT Instances used in Transaction groups only support String datatypes. Other datatypes could cause Transaction Groups groups running in previous versions of Ignition to fault.
Mode	Options for displaying values based on the item value. • Direct Value: Item value. • Hour Meter: Record the amount of time the item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The data type should be set to integer or float when using an Hour Meter regardless of the OPC Item type. ◦ On Zero: Use a zero value to accumulate time instead of a non-zero value. ◦ Retentive: Retain the Hour Meter value when it is not accumulating. ◦ Units: The time units to display. • Event Meter: Record the number or times the item Value is non-zero. The data type should be set to integer when using an Event Meter regardless of the OPC Item type. ◦ On Zero: Use a zero value to accumulate events instead of a non-zero value.

Write Target

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

Tag Reference Item Options

General

Property	Description
Name	The name of the Tag Reference Item in the group. There cannot be duplicate names within a group.
Tag Path	The path to the tag being referenced. This value is not editable except by clicking the Insert Tag button. There cannot be duplicate names within a group.
Data Type	The data type to write to into the database if this item is not read-only.

Value Mode

Property	Description
Property	Which property of the tag you want to use. • Value: Item value. • Quality: Quality code of the tag (192 = GOOD_DATA) • Timestamp: The last time the item value changed. • Name: The SQLBridge Item Name property of this item.
Mode	Options for displaying values based on the item value. • Value: Item value • Hour Meter: Record the amount of time the item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The data type should be set to integer or float when using an Hour Meter, regardless of the item type. ◦ On Zero: Use a zero value to accumulate time instead of a non-zero value. ◦ Retentive: Retain the Hour Meter value when it is not accumulating. ◦ Units: The time units to display. • Event Meter: Record the number or times the item value is non-zero. The data type should be set to integer when using an Event Meter regardless of the item type. ◦ On Zero: Use a zero value to accumulate events instead of a non-zero value.

Write Target

Property	Description
Mode	Changes the items directional read/write option. This is only editable when the target Type is set to Database field. • Use group's mode: Inherit the Update Mode from the item's group. • OPC to DB: Only read from the OPC server and write to the database. • DB to OPC: Only read from the database and write to the OPC Server. • Bi-directional OPC wins: Read and Write to both the database and OPC Server. On group start, write database values to the OPC Server. • Bi-directional DB wins: Read and Write to both the database and OPC Server. On group start, write database values to the database.
Target Type	This is the selection for what the item will write to when the group executes. • None, read-only item: Do not write this value to the database. • Database field: Write the item value to the specified column in the database table.
Target Name	The name of the column in the database that this item will write to when the group executes. The Target Name list will populate with all the column names from the Group's target table if the Target Type is Database field.

Expression Item Options

General

Property	Description
Name	The name of the Expression Item in the group. There cannot be duplicate names within a group.
Value	The static value of this Expression Item. This will be overwritten by an Expression/SQL binding.
Data Type	The data type values are stored as.

Value Mode

Property	Description
Property	Which property of the tag you want to use. • Value: Item value. • Quality: Quality code of the expression/SQL Query (192 = GOOD_DATA) • Timestamp: The last time the item value changed. • Name: The SQLBridge Item Name property of this item.
Mode	Options for displaying values based on the item value. • Direct Value: Item value. • Hour Meter: Record the amount of time the item value is non-zero. This accumulation will reset to zero when the item value goes to zero. The data type should be set to integer or float when using an Hour Meter, regardless of the item type. ◦ On Zero: Use a zero value to accumulate time instead of a non-zero value. ◦ Retentive: Retain the Hour Meter value when it is not accumulating. ◦ Units: The time units to display. • Event Meter: Record the number of times the item value is non-zero. The data type should be set to integer when using an Event Meter regardless of the item type. ◦ On Zero: Use a zero value to accumulate events instead of a non-zero value.
Evaluation Mode	Run-always (ignore Trigger): When selected, this causes the group to evaluate at each group interval, before the trigger state is evaluated.

Property	Description
Write Target	Target type: This is the selection for what the item will write to when the group executes. • None, read-only item: Do not write this value to the database. • Database field: Write the item value to the specified column in the database table. • Other Tag: Write the Expression Item's value back to an OPC Item or Tag Reference.
Target Name	The name of the column in the database that this item will write to when the group executes. The Target Name list will populate with all the Expression Item and Tag Reference names from this Group, or the column names from the Group's target table depending on the Target Type selected.
Numeric	These are the Numeric properties for Expression Items. For a full description, see Tag Scaling Properties.
Alarming	These are the Alarming settings for the Expression Items. See Alarming Properties for a full explanation.
Expression	These are the Expression/SQL Query options for Expression Items. See Expression/SQL Properties for a full explanation.

Hour and Event Meters



INDUCTIVE
UNIVERSITY

Hour and Event Meters

[Watch the video](#)

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.

Count the Duration of a Tag Being True

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

File Edit View Project Tools Help

Project Browser

Filter

- Alarm/Notification Pipelines
- Sequential Function Charts
- Scripting
- Perspective
- Transaction Groups
 - Bi Directional
 - Event_Meter
 - Group
 - New Block Group
 - New Group_Internal**

Tag Browser

Tags UDT Definitions

Tag	Value	Data Type
_Generic_Programmable_Sim		
_Sim_New_Programmable_		
Demographics		
Writeable_Test		
BooleanTag2		Boolean
Dataset	Datas...	D
Float Tag 1	null	F

New Group_Internal

Execution Disabled Save project to apply changes Enabled Disabled Pause

Basic OPC/Group Items (1)

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

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

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

Triggered Expression Items (0)

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

Status Events

Execution Disabled

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

2. From OPC Browser or Tag Browser, drag a memory tag or OPC tag (must be a numeric data type) into the **Basic OPC/Groups Items** portion of the **Standard** Transaction Group.
3. Right-click on the boolean tag in the Transaction Group and select **Edit**. The **Edit group tag** window is displayed.

New Group_Internal

▶ Enabled

⊘ Disabled

|| Pause

Unable to Start

Save project to apply changes

Basic OPC/Group Items (2)

Item Name	Source V...	Latched ...	Mode	Target Na... ¹	Data Type	Properties
Sim/Ramp/Ramp0	N/A	N/A	---	Ramp0	Float8	
Sim/Boolean Examp...	N/A	N/A	---	Boolean_E...	Boolean	

Run-Always Expression Items

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

Triggered Expression Items (0)

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

+

New Item ▶

Edit

Write to item

Cut

Copy

Paste

Delete

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

Edit group tag

Tag

Tag Item

General

Name
Sim/Boolean Example Tag

Tag Path
[default]_Sim_/ReadOnly/ReadOnlyBoolean

Data Type
Boolean

Value Mode

Property
Value

Mode
☒ Hour meter
☐ Direct value
☐ Event meter

☐ On Zero
☐ Retentive
Units
second(s)

☐ Reset on condition:

Write target

Mode
Use group's mode

Target Type
Other tag

Target Name
Sim/Ramp/Ramp0

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

New Group_Internal ▶ Enabled ⊘ Disabled || Pause
 Unable to Start Save project to apply changes

Basic OPC/Group Items (2)

Item Name	Source V...	Latched ...	Mode	Target Na... ▾ ¹	Data Type	Properties
Sim/Ramp/Ramp0	N/A	N/A	—	Ramp0	Float8	
Sim/Boolean Examp...	N/A	N/A	—	n_/Ramp/Ramp0 ▾	Boolean	

Read Only
DB Columns
Tags
Sim/Ramp/Ra

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

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

ii. For the memory tag, select Read-only from the dropdown.

New Group_Internal ▶ Enabled ⊘ Disabled || Pause
 Unable to Start Save project to apply changes

Basic OPC/Group Items (2)

Item Name	Source V...	Latched ...	Mode	Target Na... ▾ ¹	Data Type	Properties
Sim/Ramp/Ramp0	N/A	N/A	—	Ramp0 ▾	Float8	
Sim/Boolean Examp...	N/A	N/A	—	Read Only	Boolean	

DB Columns
Tags
Sim/Boolean E

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

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

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

8. Set the boolean tag to true to the start of the Hour meter.

Event Meter

Another common scenario is to count the number of times an event occurred, where an event represents a value change to the tag. For instance, the Event meter targeting a boolean tag would count each value change from false to true. Notice that this is different from the Hour meter as it counts the amount the value changes from false to true, instead of the duration the tag value is true. This count will total the number of cycles the boolean tag has experienced.

NOTE

The data type must be set to integer when using an Event meter regardless of the OPC Item type.

Count in Response to a Tag Being True

1. From OPC Browser or Tag Browser, drag a boolean tag into the **Basic OPC/Groups Items** area of a **Standard** Transaction Group.
2. From OPC Browser or Tag Browser, drag a memory tag or OPC tag 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_/Boolea...	N/A	N/A	—	 Boolean1	Boolean	
 _Sim_New_Programmable_/Events	N/A	N/A	—	 Events	Int4	

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

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

Triggered Expression Items (0)

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

3. Right-click on the boolean tag in the Transaction Group and select **Edit**.
4. In the **Edit group tag** window, set the following:
 - Value Mode: **Event meter**
 - Target Type: **Other Tag**
 - Target Name: **_Sim_New_Programmable_/Events** (or name of the tag you are using)
5. Click **OK**.

Edit group tag

Tag

Tag Item

- General

Name
_Sim_New_Programmable_/Boolean1

Tag Path
[default]_Sim_New_Programmable_/Boolean1

Data Type
Boolean

- Value Mode

Property
Value

Mode

☐ Direct value
☐ Hour meter
☒ Event meter

☐ On Zero ☐ Retentive Units: second(s)

☐ Reset on condition: > 0

- Write target

Mode
Use group's mode

Target Type: Other tag
Target Name: _Sim_New_Programmable_/Events

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

Event-Meter-2 ▶ Enabled ⏻ Disabled ⏸ Pause
 Execution Disabled Save project to apply changes

Basic OPC/Group Items (2)

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

Read Only
DB Columns
Tags
_Sim_New_Pr

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
-----------	--------------	---------------	-------------	-----------	------------

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

Reset an Hour or Event Meter Based on a Condition



INDUCTIVE
UNIVERSITY

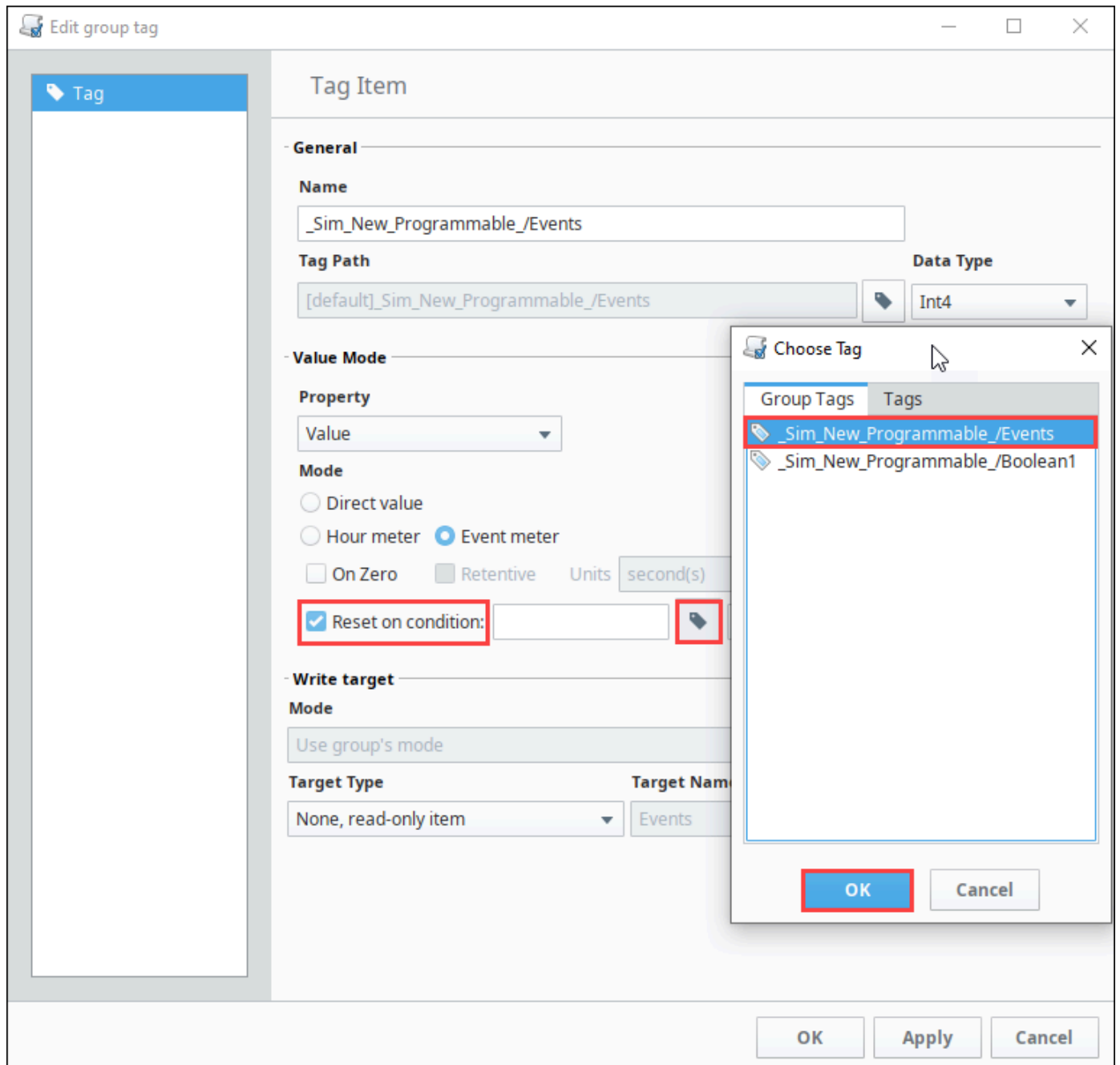
**Resetting Hour and Event
Meters**

[Watch the video](#)

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

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

4. Click **OK**.



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

Edit group tag

Tag

Tag Item

General

Name:

Tag Path: Data Type:

Value Mode

Property:

Mode:

☐ Direct value

☐ Hour meter

☒ Event meter

☐ On Zero ☐ Retentive Units:

☒ Reset on condition:

Write target

Mode:

Target Type: Target Name:

OK Apply Cancel

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

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](#).

Standard Group

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

Bi Directional

Execution Disabled Save project to apply changes

Enabled

Disabled

Pause

Basic OPC/Group Items (8)

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

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

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

Triggered Expression Items (0)

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

ActionTriggerOptions

Execution Scheduling:

TimerSchedule

1second(s)

Update mode:
Bi-directional OPC wins

Data source:
<Default>

Table name:
group_table

Automatically create table

Use custom index column: group_table_ndx

Store timestamp to: t_stamp

Store quality code to: quality_code

Delete records older than:
1day(s)

Table action

insert new row

update/select

first

last

custom

key/value pair

Historical Group

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

New Historical Group

Execution Disabled Save project to apply changes

Enabled

Disabled

Pause

Basic OPC/Group Items (10)

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

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

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

Triggered Expression Items (0)

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

ActionTriggerOptions

Execution Scheduling:

TimerSchedule

1second(s)

Data source:
<Default>

Table name:
group_table

Automatically create table

Use custom index column: group_table_ndx

Store timestamp to: t_stamp

Store quality code to: quality_code

Delete records older than:
1day(s)

Block Group

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

New Block Group

Execution Disabled Save project to apply changes

► Enabled

⏸ Disabled

⏸ Pause

Item View

Block View

Block Items (3)

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

Basic OPC/Group Items (0)

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

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

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

Triggered Expression Items (0)

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

⚡ Action

▶ Trigger

⚙ Options

Execution Scheduling:

⬢ Timer

⬢ Schedule

1second(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

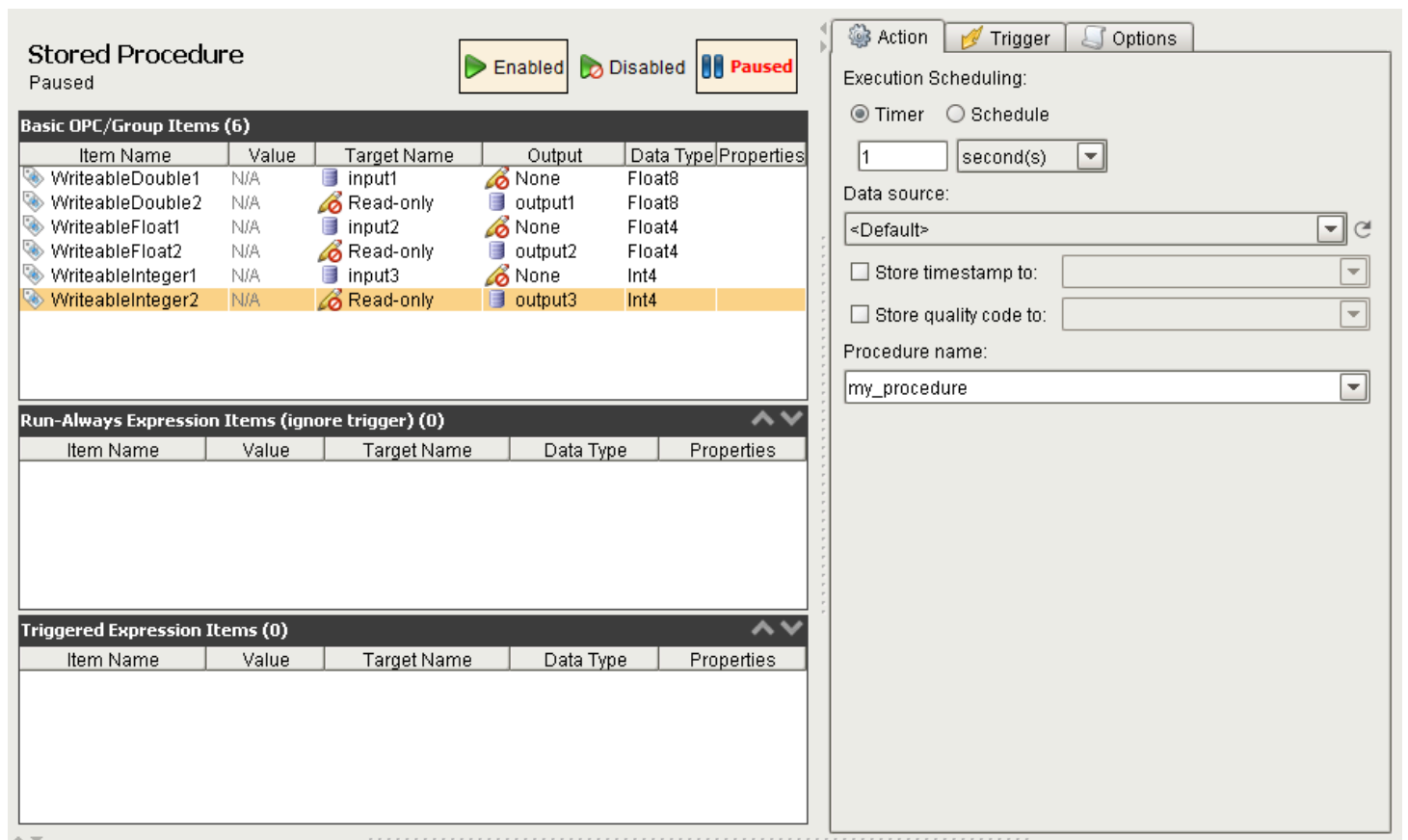
☐ Delete records older than: 1day(s)

Table action

⬢ insert new block

Stored Procedure Group

The Stored Procedure Group is used in a similar way as the Historical Group except instead of selecting a target table, you select a target Stored Procedure and use the tags as inputs and outputs. The only major requirement that makes this group different is that a stored procedure must already exist in your database for this group to work.



Configuring Transaction Group for OPC to OPC Interaction

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

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

Project Browser

Filter

Project Properties

Alarm Notification Pipelines

Sequential Function Charts

Scripting

Perspective

Transaction Groups

opc to opc

Vision

Named Queries

Reports

Tag Browser

Tag

Value

Data Type

Traits

Tags

Data Types

tag1 OPC

tag2 OPC

System

All Providers

opc to opc

Running

Enabled

Disabled

Pause

Basic OPC/Group Items (2)

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

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

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties
-----------	--------------	---------------	-------------	-----------	------------

Triggered Expression Items (0)

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties
-----------	--------------	---------------	-------------	-----------	------------

Action

Trigger

Options

Execution Scheduling:

Timer

Schedule

1

second(s)

Update mode:

OPC to DB

Data source:

DB

Table name:

group_table

Automatically create table

Use custom index column:

group_table_ndx

Store timestamp to:

t_stamp

Store quality code to:

quality_code

Delete records older than:

1

day(s)

Table action

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

Block Group



INDUCTIVE
UNIVERSITY

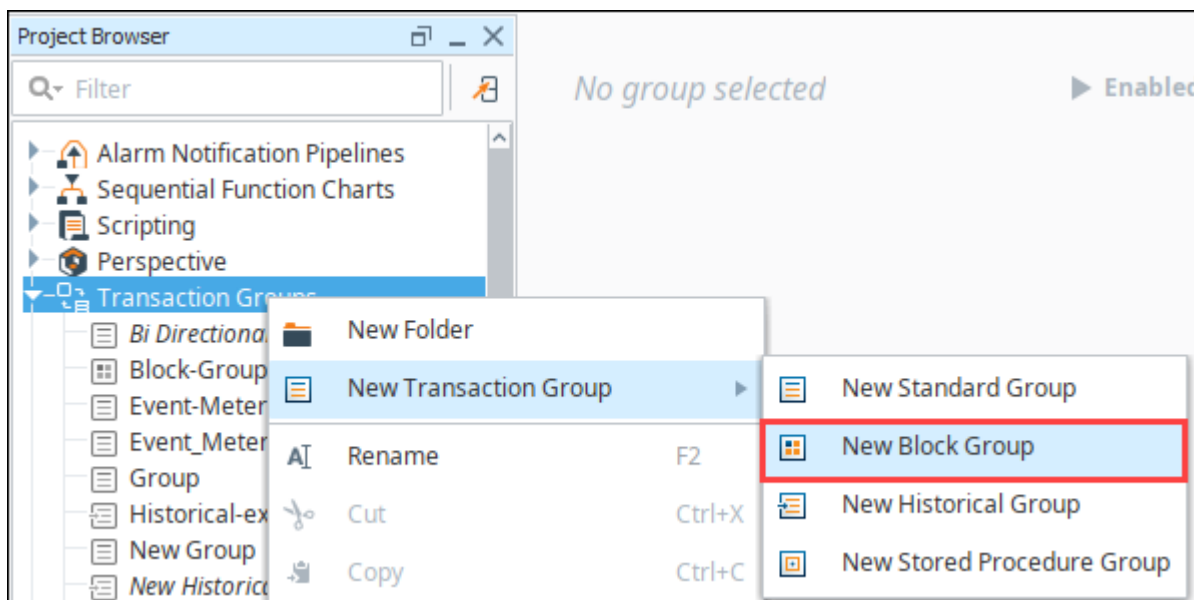
Block Group Type

[Watch the video](#)

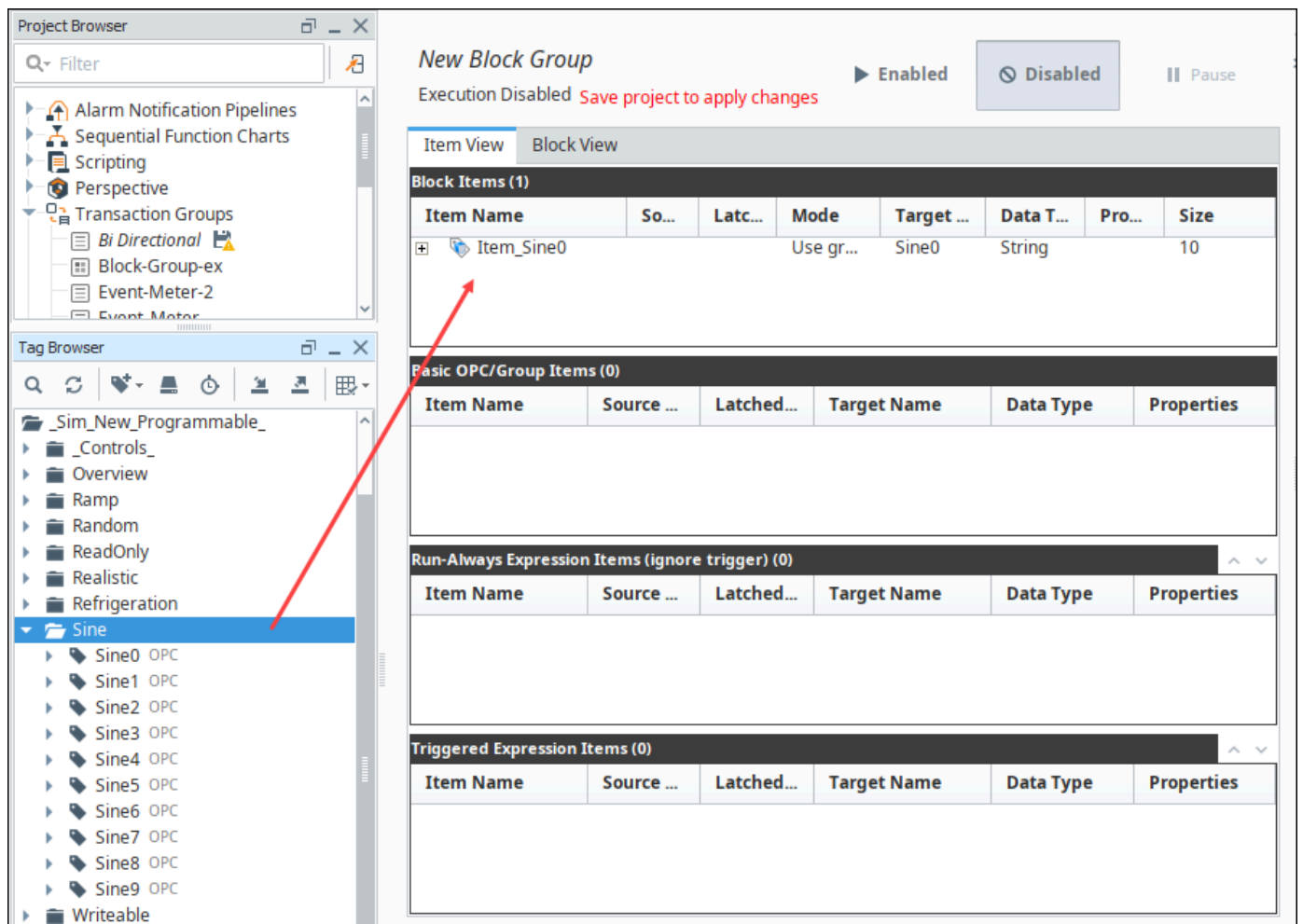
The Block group is a type of transaction group that stores data vertically. Whereas, a Standard group stores the information horizontally in a single row. Block groups share many of the same features as the Standard group. They can be bidirectional, insert into a database, or simply update the database. 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.

Create a Block group

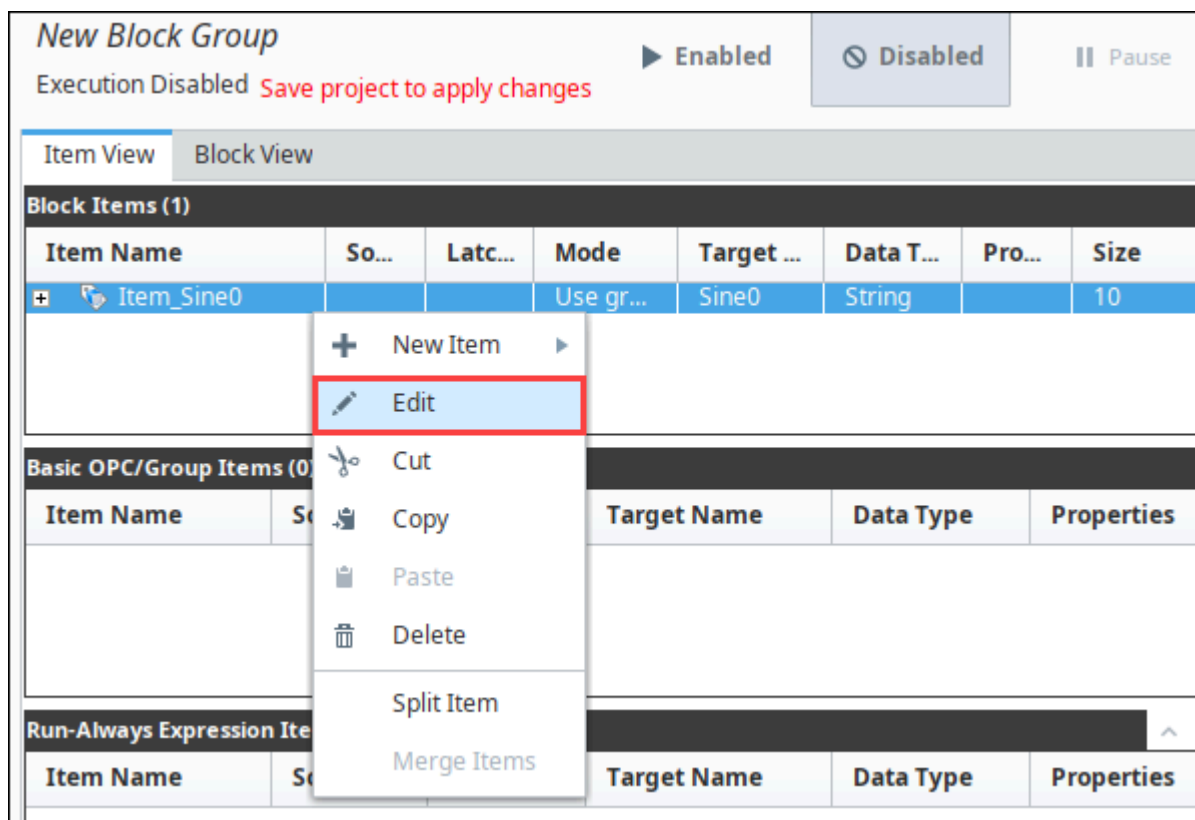
1. In the Project Browser, right-click on Transaction Groups and select **New Transaction Group > New Block Group**.



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**.

Edit group tag

Block Item

Block Item

Items

Numeric

Alarming

General

Name: Block Group Sine

Data Type: Float8

Value Mode

Property: Value

Mode:

☒ Direct value

☐ Hour meter ☐ Event meter

☐ On Zero ☐ Retentive Units: second(s)

☐ Reset on condition: [] [] > [] 0

Write target

Target Type: Database field

Target Name: Sine0

OK Apply Cancel

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

⚡ Action

▶ Trigger

⚙ Options

Execution Scheduling:

☒ Timer
 ☐ Schedule

Update mode:

Data source:

Table name:

☒ Automatically create table
 ☒ Automatically create rows

☐ Use custom index column:

☒ Store timestamp to:

☐ Store quality code to:

☐ Store row id to:

☐ Store block id to:

☐ Delete records older than:

Table action

☒ insert new block
 ☐ insert changed rows
 ☐ update/select

☒ first
 ☐ last
 ☐ custom

Where:

7. Select the group, and click **Enabled**.

8. **Save** the project to start the group.

DB to OPC Mode with Custom Where Clause

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

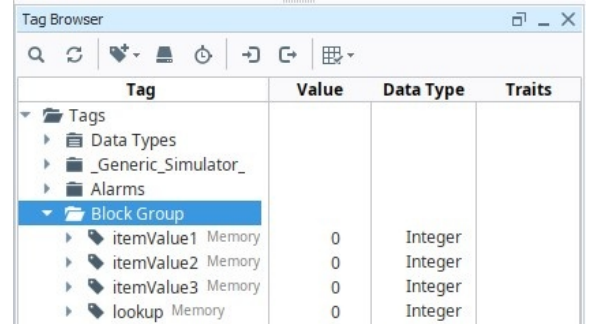
This is a great way to retrieve multiple datapoints that are stored in a tall format on a database table, ideally when you're looking to retrieve multiple sequential rows. For example a table with the following content, a single block item targeting the "itemValue" column, and a "lookup" tag or OPC item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

Table Structure, Block Item, and Tags

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

Our Tags (including lookup)



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 our condition. In this case, we'll say we want results from our table starting a value greater than our lookup value. Using the table specified above, we could write the following condition:

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

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

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

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

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

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

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

Values on our Tags

Block Group

itemValue1

Memory

7,000,000

Integer

itemValue2

Memory

50,000

Integer

itemValue3

Memory

600,000

Integer

lookup

Memory

6

Integer

Items in the Group

Item View

Block View

Block Items (1)

Item Name	Source Val...	Latched V...	Mo...	Target Name	Da...	...	...
itemValue [default]Block Group/itemValue1 [default]Block Group/itemValue2 [default]Block Group/itemValue3	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](#).

Recipe Group



INDUCTIVE
UNIVERSITY

Recipe Group

[Watch the video](#)

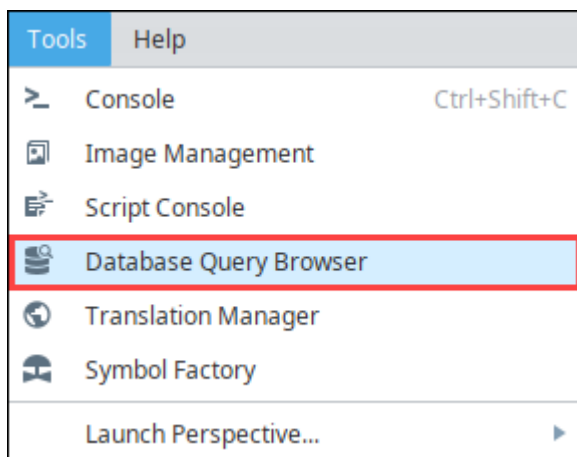
You can use Transaction Groups to create a recipe management system which will pull recipe information from the database and push it to the PLC when requested. With this system, the Transaction Group is what queries the database rather than writing scripts to handle it all.

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

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

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

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

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

id	recipe_name	setpoint1	setpoint2
1	The First Recipe	34.7	54.1
2	The Wrong Recipe	12.8	42.3
3	The Best Recipe	65.7	95.1
4	The Other Recipe	49.8	112.2

Create the Transaction Group

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

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

Tag Browser

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

Recipe Group

Execution Disabled Save project to apply changes ► Enabled ⏏ Disabled ⏏ Pause

Basic OPC/Group Items (4)						
Item Name	Sou...	La...	Mode	Target ...	Data ...	Prop...
Float Tag 1	N/A	N/A	Use g...	Float_T...	Float4	
Float Tag 2	N/A	N/A	Use g...	Float_T...	Float4	
Integer Tag	N/A	N/A	Use g...	Integer...	Int4	
String Tag	N/A	N/A	Use g...	String_T...	Int4	

Run-Always Expression Items (ignore trigger) (0)					
Item Name	Sour...	Latch...	Target Name	Data Type	Properti...

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

Basic OPC/Group Items (4)						
Item Name	Source ...	Latche...	Mode	Target Name	Data Type	Properties
Integer Tag	N/A	N/A	---	Read-only	Int4	
String Tag	N/A	N/A	---	recipe_name	String	
Float Tag 1	N/A	N/A	---	setpoint1	Float4	
Float Tag 2	N/A	N/A	---	setpoint2	Float4	

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

⚡ Action
▶ Trigger
⚙ Options

Execution Scheduling:

☒ Timer
 ☐ Schedule

Update mode:

DB to OPC

Data source:

<Default>

Table name:

recipes

☒ Automatically create table

☐ Use custom index column: recipes_ndx

☒ Store timestamp to: t_stamp

☐ Store quality code to: quality_code

☐ Delete records older than:

Table action

☐ insert new row
 ☒ update/select
 ☐ first
 ☐ last
 ☐ custom
 ☒ key/value pairs

Column	Value
id	Integer Tag

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

⚡ Action

▶ Trigger

⚙ Options

☐ Only evaluate when values have changed.

Tags to watch for change:

All tags ▼

Select tags

☒ Execute this group on a trigger

Trigger on item:

Integer Tag ▼

☐ Only execute once while trigger is active

☐ Reset trigger after execution

☐ Prevent trigger caused by group start

Trigger conditions:

☐ is != 0 (or true)

☐ is = 0 (or false)

☐ is active: > ▼ 0

non-active: <= ▼ 1

☒ Active on value change

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

Update or Insert Group



INDUCTIVE
UNIVERSITY

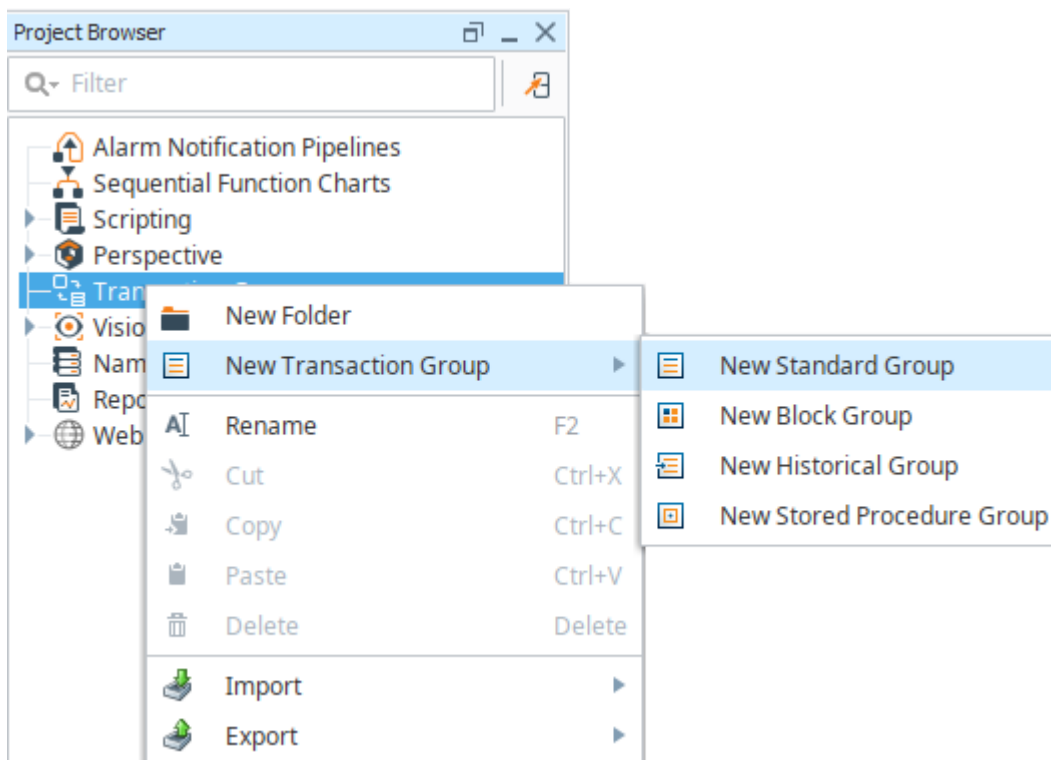
Update or Insert Group

[Watch the video](#)

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 a New Row into the Database

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



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

Update Group
Execution Disabled Save project to apply changes

Basic OPC/Group Items (5)

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

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

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

Triggered Expression Items (0)

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

Status Events

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

Update Group
Execution Disabled Save project to apply changes

Basic OPC/Group Items (5)

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

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

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

Triggered Expression Items (0)

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

- 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

Update mode:

Data source:

Table name:

☒ Automatically create table

☐ Use custom index column:

☒ Store timestamp to:

☐ Store quality code to:

☐ Delete records older than:

Table action

☐ insert new row

☒ update/select

☐ first ☐ last ☐ custom ☒ key/value pairs

Column	Value
	<i>null</i>

  ☐ Insert row when not present

6. Click the **Add** () icon.

- i. For the **Column**, select the database table ID column.
- ii. For the **Value** column, select the read-only tag.
- iii. 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.

Trigger Options



INDUCTIVE
UNIVERSITY

Trigger - On Value Change

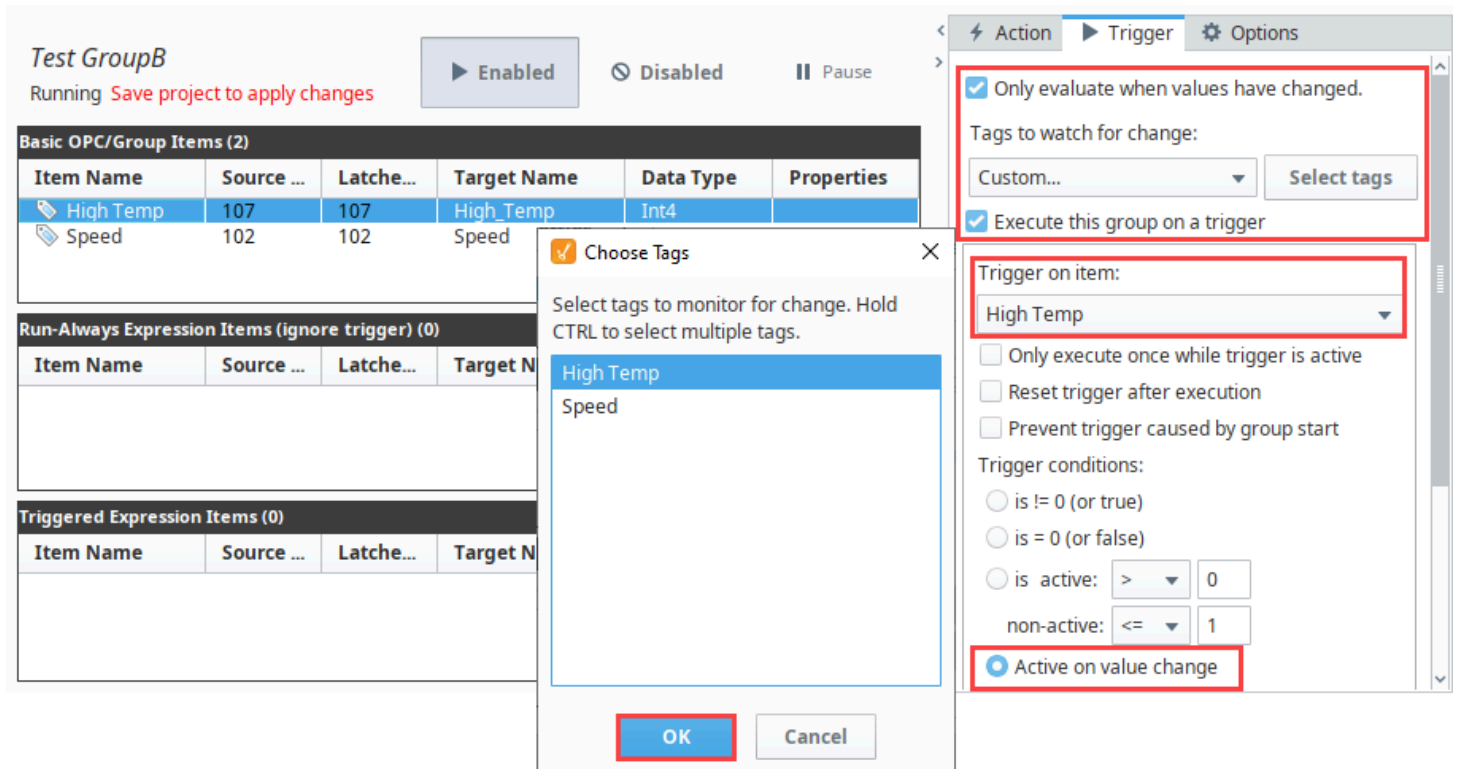
[Watch the video](#)

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.



Execute while Condition is True

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

1. Create a Transaction Group, and drag a numeric or boolean tag into the **Basic OPC/Group Items** section.
2. From the **Target Name** column dropdown, select **Read Only** if you do not want the trigger value to be written to the database.
3. Go to the **Trigger** tab, and select the **Execute this group on a trigger** checkbox.
4. In the **Trigger conditions** area, set the trigger conditions which will determine under what condition the group executes.
5. **Save** the project to start the Transaction Group.

The screenshot displays the 'Test GroupA' configuration window. The 'Trigger' tab is selected, showing the 'Execute this group on a trigger' checkbox checked. The 'Trigger on item' dropdown is set to 'High Temp'. The 'Trigger conditions' section shows 'is active' selected with a value of 100, and 'non-active' with a value of 99. The 'Basic OPC/Group Items (2)' table shows 'High Temp' with a source value of 105 and a latched value of 105. The 'Target Name' dropdown for 'High Temp' is open, showing 'Read Only' selected. The 'Run-Always Expression Items (ignore trigger) (0)' and 'Triggered Expression Items (0)' tables are empty.

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

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

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

Execute on a Rising Edge

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

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

Group S

Running

▶ Enabled

⊘ Disabled

⏸ Pause

Basic OPC/Group Items (4)

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

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

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

Triggered Expression Items (0)

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

⚡ Action

▶ Trigger

⚙ Options

☒ Only evaluate when values have changed.

Tags to watch for change:

Custom...

Select tags

☒ Execute this group on a trigger

Trigger on item:

BooleanTag2

☒ Only execute once while trigger is active

☐ Reset trigger after execution

☐ Prevent trigger caused by group start

Trigger conditions:

☐ is != 0 (or true)

☒ is = 0 (or false)

☐ is active: > 0

☐ non-active: <= 1

☐ Active on value change

Reset Trigger

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

To reset the trigger after execution:

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

testHistorical
Running

▶ Enabled
⏏ Disabled
⏸ Pause

Basic OPC/Group Items (3)

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

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

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

Triggered Expression Items (0)

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

⚡ Action ▶ Trigger ⚙ Options

☐ Only evaluate when values have changed.

Tags to watch for change:

All tags ▼
Select tags

☒ Execute this group on a trigger

Trigger on item:

testBoolean ▼

☐ Only execute once while trigger is active

☒ Reset trigger after execution

☐ Prevent trigger caused by group start

Trigger conditions:

☒ is != 0 (or true)
☐ is = 0 (or false)
☐ is active: > ▼ 0
☐ non-active: <= ▼ 2
☐ Active on value change

☐ Write handshake on success

Set: ▼

Handshakes

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

To set handshake values for alerting the user:

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

Group Z

Running

► Enabled

⊘ Disabled

⏸ Pause

Basic OPC/Group Items (2)

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

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

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

Triggered Expression Items (0)

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

⚡ Action ▶ Trigger ⚙ Options

☒ Execute this group on a trigger

Trigger on item:

BooleanTag ▼

- ☐ Only execute once while trigger is active
- ☐ Reset trigger after execution
- ☐ Prevent trigger caused by group start

Trigger conditions:

☒ is != 0 (or true)

☐ is = 0 (or false)

☐ is active: > ▼ 0

non-active: <= ▼ 1

☐ Active on value change

☒ Write handshake on success

Set: WriteableInteger5 ▼

To value: 1

☒ Write handshake on failure

Set: WriteableInteger5 ▼

To value: 2

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.

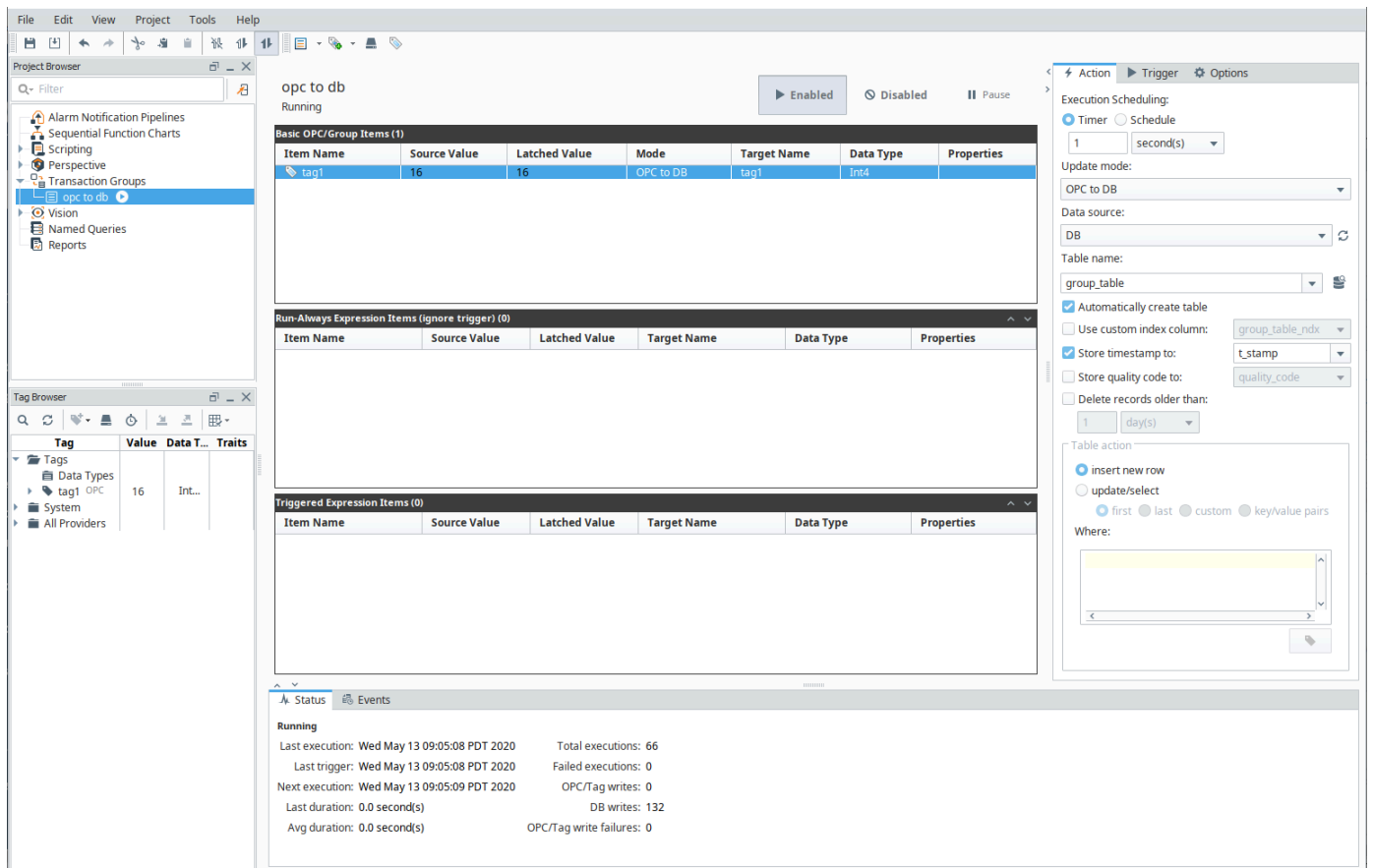
The different Update Modes:

- 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.

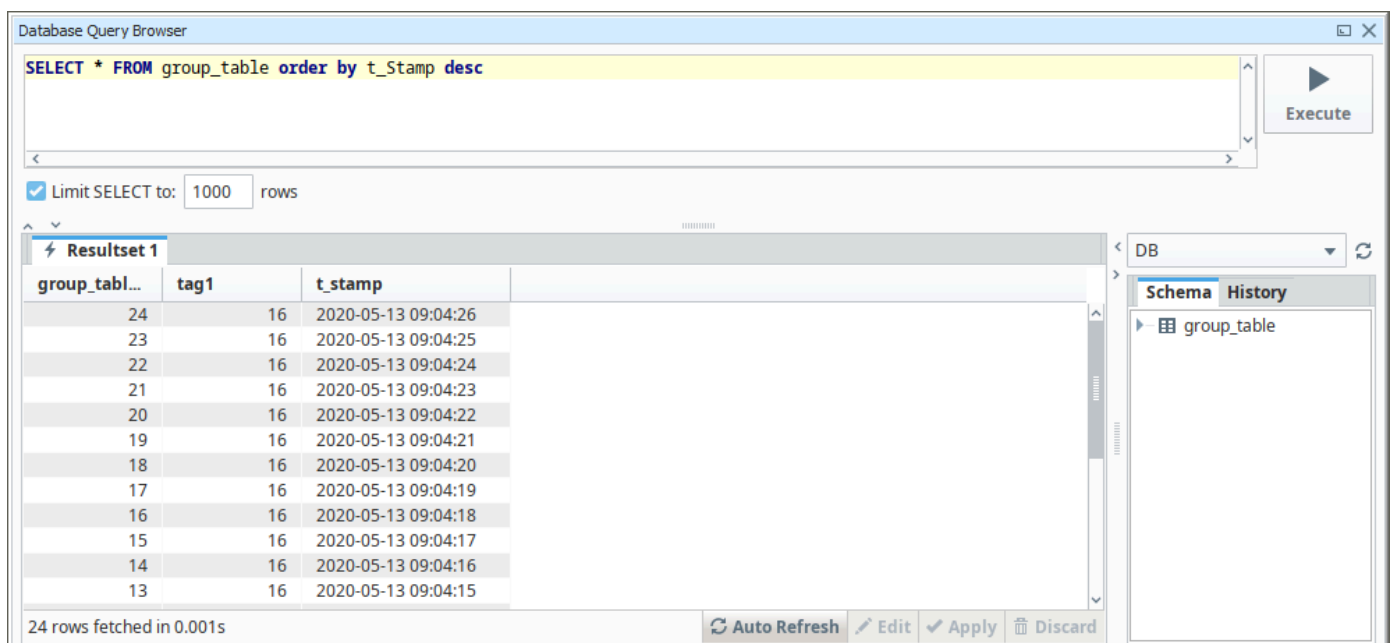
OPC to DB

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.



This configuration will allow for tag1's value to be stored into a database table called **group_table** every 1 second to a column called tag1. We can see this working now through the [Database Query Browser](#).



DB to OPC

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

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

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

tag1 Value: 22

The screenshot shows the Tag Browser on the left with tag1 at value 22. The Transaction Group configuration in the center shows the Table Action set to 'update/select' with the 'last' option selected. The Database Query Browser on the right shows a query: `SELECT * FROM group_table order by t_stamp desc`. The result set shows a single row with tag1 value 22.

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties
group_table	22		tag1	INT	

tag1 Value: 29

The screenshot shows the Tag Browser on the left with tag1 at value 29. The Transaction Group configuration in the center shows the Table Action set to 'update/select' with the 'last' option selected. The Database Query Browser on the right shows a query: `SELECT * FROM group_table order by t_stamp desc`. The result set shows a single row with tag1 value 29.

Item Name	Source Value	Latched Value	Target Name	Data Type	Properties
group_table	29		tag1	INT	

Bi-directional OPC Wins

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

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

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

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

The screenshot displays the Ignition configuration interface for a Transaction Group. The **Basic OPC/Group Items (1)** table shows the configuration for tag1:

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

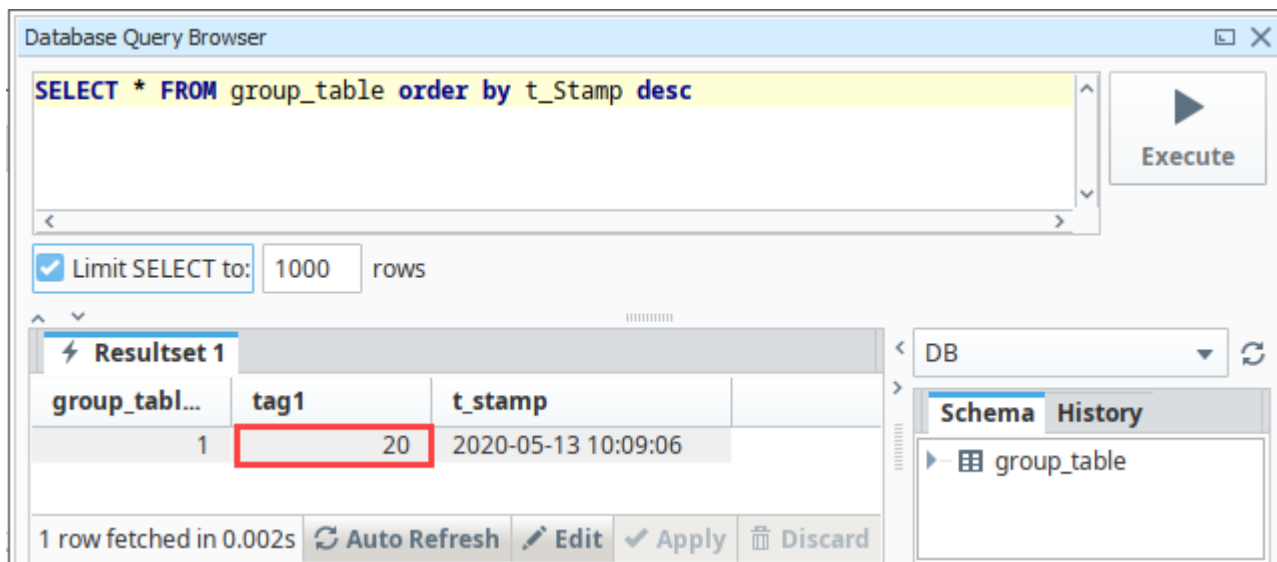
The **Database Query Browser** window shows the SQL query: `SELECT * FROM group_table order by t_stamp desc`. The result set shows one row:

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

The **Tag Browser** shows the tag1 tag with a value of 20 and data type of Integer.

The **Table action** section on the right is configured with the following settings:

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



Bi-directional DB Wins

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

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

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

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

Perspective

Transaction Groups

db to opc

Vision

Named Queries

Reports

Tag

Value

Data Type

Traits

Tags

Data Types

tag1 OPC

10

Integer

System

All Providers

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

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

Database Query Browser

SELECT * FROM group_table order by t_stamp desc

Limit SELECT to: 1000 rows

ResultSet 1

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

1 row fetched in 0.001s

Auto Refresh

Edit

Apply

Discard

Schema History

group_table

Update mode:

Bi-directional DB wins

Data source:

DB

Table name:

group_table

☒ Automatically create table

☐ Use custom index column:

group_table_ndx

☒ Store timestamp to:

t_stamp

☐ Store quality code to:

quality_code

☐ Delete records older than:

1

day(s)

Table action

☐ insert new row

☒ update/select

☐ first

☒ last

☐ custom

☐ key/value pairs

Where:

File Edit View Project Tools Help

Project Browser

Filter

Alarm Notification Pipelines

Sequential Function Charts

Scripting

Perspective

Transaction Groups

db to opc

Vision

Named Queries

Reports

Tag

Value

Data Type

Traits

Tags

Data Types

tag1 OPC

20

Integer

System

All Providers

db to opc

Running

Enabled

Disabled

Pause

Basic OPC/Group Items (1)

Item Name	Source Value	Latched Value	Mode	Target Name	Data Type	Properties
tag1	20	20	Bi-directional DB wins	tag1	Int4	

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

Item Name

Source Value

Latched Value

Target Name

Data Type

Properties

Triggered Expression Items (0)

Item Name

Source Value

Latched Value

Target Name

Data Type

Properties

Status

Events

Running

Last execution: Wed May 13 10:28:41 PDT 2020

Total executions: 4

Last trigger: Wed May 13 10:28:41 PDT 2020

Failed executions: 0

Next execution: Wed May 13 10:28:42 PDT 2020

OPC/Tag writes: 1

Last duration: 0.0010 second(s)

DB writes: 0

Avg duration: 0.0 second(s)

OPC/Tag write failures: 0

Database Query Browser

SELECT * FROM group_table order by t_stamp desc

Limit SELECT to: 1000 rows

ResultSet 1

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

1 row fetched in 0.001s

Auto Refresh

Edit

Apply

Discard

Schema History

group_table

Action

Trigger

Options

Execution Scheduling:

Timer

Schedule

1

second(s)

Update mode:

Bi-directional DB wins

Data source:

DB

Table name:

group_table

☒ Automatically create table

☐ Use custom index column:

group_table_ndx

☒ Store timestamp to:

t_stamp

☐ Store quality code to:

quality_code

☐ Delete records older than:

1

day(s)

Table action

☐ insert new row

☒ update/select

☐ first

☒ last

☐ custom

☐ key/value pairs

Where: