

Ignition 7.9 User Manual — Database Connections and Tags

Generated 2026-10-12 from
<https://www.docs.inductiveautomation.com/docs/7.9/>

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Database Connections

How are Databases Used in Ignition?

While connecting to a database is not required for basic status and control functionality, it can dramatically increase the possibilities that the system offers. There are a few places where databases are used in Ignition, such as historical data logging, reporting, storing alarm logs, and as your tags storage.

Historical Data Logging

Logging data for historical analysis, either through [Tags Historian](#) or with the [SQL Bridge module](#), requires a database connection. Databases are great at handling historical data, and by using a standard relational database your data is stored in an open format that can be used in many ways.

Reports, Graphs, and Charts

The Vision module makes it easy to present data stored in databases in a variety of ways. You can [quickly create charts](#) that show performance over time, locate anomalies, detect trends, and more. Furthermore, it's important to remember that it is possible to pull data from any database that Ignition is connected to, even if the data wasn't placed there by Ignition. This means you can tie in data from other sources or areas of your company, such as pulling in inventory and staff information, as well.

Storing Alarm Logs

[Store alarm information historically](#) and examine it later for patterns or trouble spots.

Database-Driven Tags

You can use a SQL database as your [Tags repository](#). Any other Ignition system with access to the database will be able to share and contribute tags, allowing you to create highly integrated distributed systems. For example, multiple plant sites can use Tags to report current status over a secure network connection to a central corporate headquarters.

Getting Started with Databases

The first step in using a database with Ignition is to identify a database server. Many companies already have database servers maintained by their IT departments. If you do not, or wish to set up your own database server for Ignition, the Supported Databases section below offers some advice on choosing a database vendor.

Once you've identified a server, all you need to do is [create a connection](#) to that server to get up and running.

Database Connections

Name	Description	JDBC Driver	Translator	Status	
MySQL		MySQL ConnectorJ	MYSQL	Valid	delete edit

[→ Create new Database Connection...](#)

Note: For details about a connection's status, see the [Database Connection Status](#) page.

Supported Databases in Ignition

Ignition has been tested with the following databases, and can connect to them directly after installation. You can connect to other databases by installing additional JDBC drivers (the Java database connection specification), which are often provided by database vendors.

Full Support

Database	Version
MySQL	5.0+ for full support. Ignition can connect to 4.x, but many features such as Tags are not tested. MySQL 8.0 New in 7.9.9 Ignition 7.9.9+ contains an updated driver that will connect to MySQL 8.0 installations. Note that upgrading Ignition does not update the JDBC driver, so pre-existing installations will need to manually update their JDBC driver.
Microsoft SQL Server	2005, 2008, 2012, 2014, 2016 (full and express editions). Ignition can connect to 2000, but has not been fully tested
Oracle	10g, 11g, 12c (full and express). The letters stand for "grid" and "cloud"
PostgreSQL	8.0+
Firebird	All versions.
IBM DB2	9.5+

Limited Support

Database	Version
Other JDBC drivers	Due to variances in databases, some features may not work fully through other non-tested JDBC drivers. However, it is usually possible to get full functionality though the careful use of the database translator feature. For example, the JDBC driver for MariaDB could be downloaded and added to Ignition.

Installing and Connecting to a Database

Once you've identified a server, all you need to do is create a connection to that server to get up and running. See the [Installing Databases](#) and [Connecting to Databases](#) sections for details about how to install and connect to different databases through Ignition.

If we don't already have a connector for your database type, you can [simply add it in](#) yourself.

Database Drivers and Translators

What is JDBC?

JDBC stands for the Java DataBase Connectivity API. It is a standardized way for Java-based applications to interact with a wide range of databases and data sources. A JDBC Driver enables Ignition to connect to, and use data from, a particular database system.

JDBC in Ignition

Ignition, being a Java-based application, leverages JDBC in order to connect to a variety of data sources. This enables Ignition to offer a standardized set of functionality on a wide range of different systems and databases. This includes not only commonly-used databases such as MySQL, Microsoft SQL Server, and Oracle, but additionally other lesser-known systems as well, provided the manufacturer offers a JDBC driver for the system.

Controller | Master

USER MANUAL | SUPPORT

admin | Sign Out

Launch Designer

Ignition
by inductive automation

HOME | STATUS | CONFIGURE

Search...

License Incomplete ? 0:00:00

Reset Trial | View Modules

SYSTEM

Overview

Backup/Restore

Licensing

Modules

Projects

Redundancy

Gateway Settings

NETWORKING

Gateway Network

Email Settings

SECURITY

Auditing

Users, Roles

Service Security

Security Zones

DATABASES

Connections

Drivers

Store and Forward

ALARMING

General

Journal

Notification

Database Drivers & Settings

JDBC Drivers | Translators

Name	Description	Driver Type	Default Translator	JAR File(s)		
Firebird JDBC Driver	The Jaybird JDBC driver for Firebird	FIREBIRD	FIREBIRD	• jaybird-full-2.1.6.jar	delete	edit
IBM DB2	The official IBM DB2 JDBC Driver.	DB2	DB2	• db2jcc4.jar	delete	edit
Microsoft SQL Server JDBC Driver	The Microsoft SQL Server JDBC Driver is a Java Database Connectivity (JDBC) 4.0 compliant driver.	Microsoft SQL Server	MSSQL	• sqljdbc4.jar	delete	edit
MySQL Connector/J	The official MySQL JDBC Driver, Connector/J.	MySQL	MYSQL	• mysql-connector-java-5.1.23-bin.jar	delete	edit
Oracle JDBC Driver	The Oracle Database JDBC driver.	Oracle	ORACLE	• ojdbc6.jar	delete	edit
PostgreSQL JDBC Driver	The official PostgreSQL JDBC Driver.	PostgreSQL	POSTGRES	• postgresql-9.0-801.jdbc4.jar	delete	edit

→ Create new JDBC Driver...

JDBC vs. ODBC

JDBC differs from ODBC (Microsoft's OpenDataBase Connectivity standard) primarily in the fact that JDBC is written in Java, and thus can be used without modification in cross-platform environments. Additionally, whereas ODBC is a complex standard that is becoming technically out-dated, JDBC is a modern, clean specification for cross-vendor database access.

Can I Connect Using ODBC?

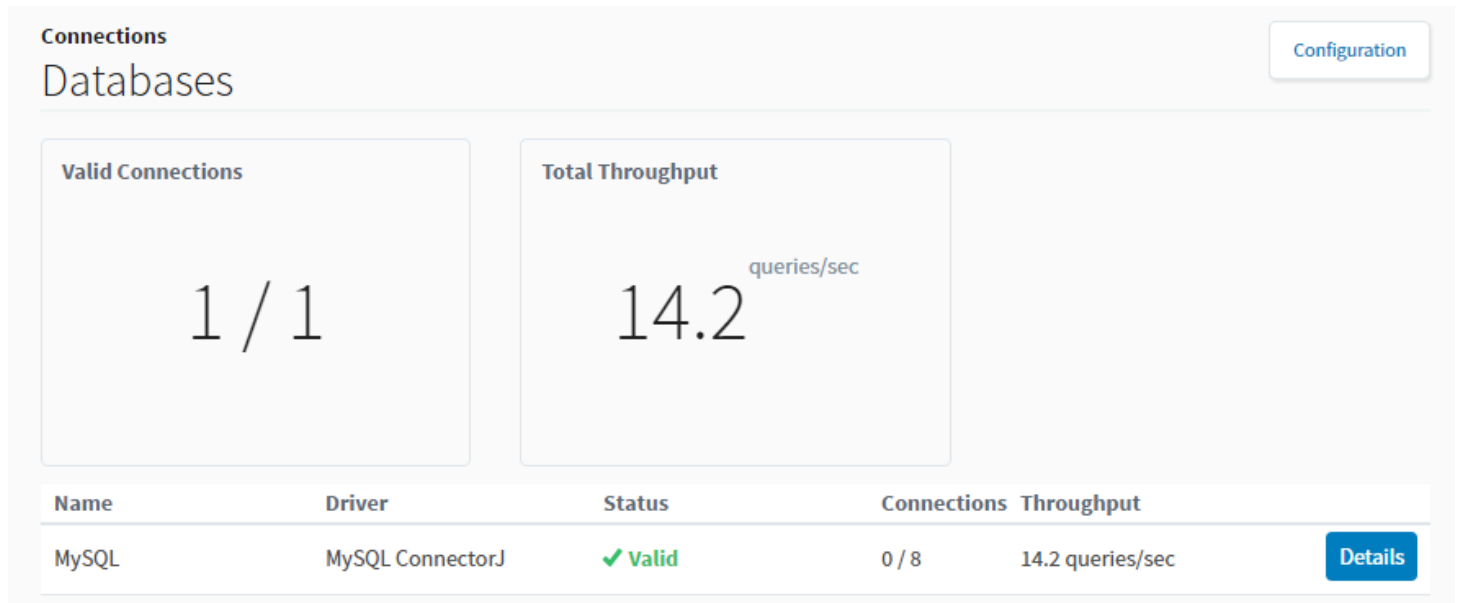
While it is indeed possible to connect to an ODBC data source using the JDBC-ODBC bridge, this is generally not advised. The bridge is designed to offer a minimal amount of functionality, and is considered a "transitional solution", meaning that it should only be used when JDBC is not available. In other words, if a JDBC option is available, ODBC should not be used.

Since most commercial databases offer JDBC drivers, transition is usually as simple as recreating your database connections inside of Ignition. The lack of a JDBC connection inside of Ignition does not necessarily indicate that JDBC isn't available for your particular database. Licensing restrictions sometime prevent the inclusion of drivers with 3rd-party software. Therefore, before using ODBC, due diligence should be taken to verify that no JDBC solution is available.

Monitoring Connection Status

The state or status of a database can be monitored from the **Status** section of the Gateway, under **Connections > Databases**. The status panels show the current state and a fault message, if applicable, or throughput statistics if the connection is active.

When a connection is not available, it is re-tested every 10 seconds, and the status is updated.



Why Use Databases?

Providing access to relational databases is at the heart of the Ignition platform. Ignition can connect to any SQL database that has a JDBC driver, though depending on the database's capabilities, some features may not be available. By connecting Ignition to one or more SQL databases, you can query existing data, update data, store historical information, and more.

Ignition can perform many tasks without the use of a database. For instance, the Vision and OPC-UA modules let you create powerful HMI status and control screens, or Tags can be used to generate alarms that can be sent over email. However, tightly integrated database access is a key feature that makes Ignition stand out from its competitors.

Modern relational databases offer amazing storage and querying capabilities with great performance at a price that is incomparable to older legacy historians. While it is true that historians still have a place in the industry, for most applications relational SQL databases not only suffice, but offer much more than what was previously available. Using SQL, you can store and track production information with ease. However, you can also correlate that data to who was on shift, previous runs, downtime, inventory levels and more, naturally and easily. Make the data available to more people using the Vision module's web-launch clients, or integrate the data directly into your company's internal or

external website. SQL databases are at the heart of the web and modern corporate IT systems, and now thanks to Ignition, the plant floor as well.

simple_histo...	accumulator...	ambientHum	ambientTemp	receiverLevel	valveDischar...	valveKing	t_stamp	quality_code	
773	76	53.56	93.29	19	1		0 Nov 30, 2015 9:...	192	▲
774	82	54.4	94.55	10	1		0 Nov 30, 2015 9:...	192	
775	80	54.97	100	14	1		0 Nov 30, 2015 9:...	192	■
776	80	54.97	100	14	1		0 Nov 30, 2015 9:...	192	
777	84	47.2	99.54	7	1		0 Nov 30, 2015 9:...	192	
778	82	44.84	97.14	5	1		0 Nov 30, 2015 9:...	192	
779	92	44.15	100	13	1		0 Nov 30, 2015 9:...	192	
780	48	34.48	66.49	57	0		1 Nov 30, 2015 1...	192	
781	55	39.97	76.34	59	0		1 Nov 30, 2015 1...	192	▼

Central Management

Database connections have a lot of rules for connecting and querying them, but Ignition simplifies all that by creating database connections in the Gateway instead of in the clients. This means from one central location, you can manage all your database connections. One massive problem with databases can be setting up a connection because everything is based on IP Addresses and computer names. Since Ignition connects through the Gateway, all clients have access to your databases and you don't have to worry about adding individual clients to the database rules. Just set up the gateway and launch clients wherever you'd like.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Installing Databases

Why Install a Database?

A lot of additional functionality becomes simple or is only accessible when Ignition is connected to a database. Storing Historical data, storing notes or files, and creating dynamic lists to name a few. It is important to note that Ignition does not install any databases for you. There are many types that you can connect to, but you need to choose the database that is best for you. Installing your own database means you have complete control over it, anything Ignition adds to it can be accessed by another program easily.

You can install as many database systems as you like, and each of them allow your to create as many schemas (or groups of data) as you want. You get to decide where your database is installed, or where you want to install all of them.

Which Database Should You Use?

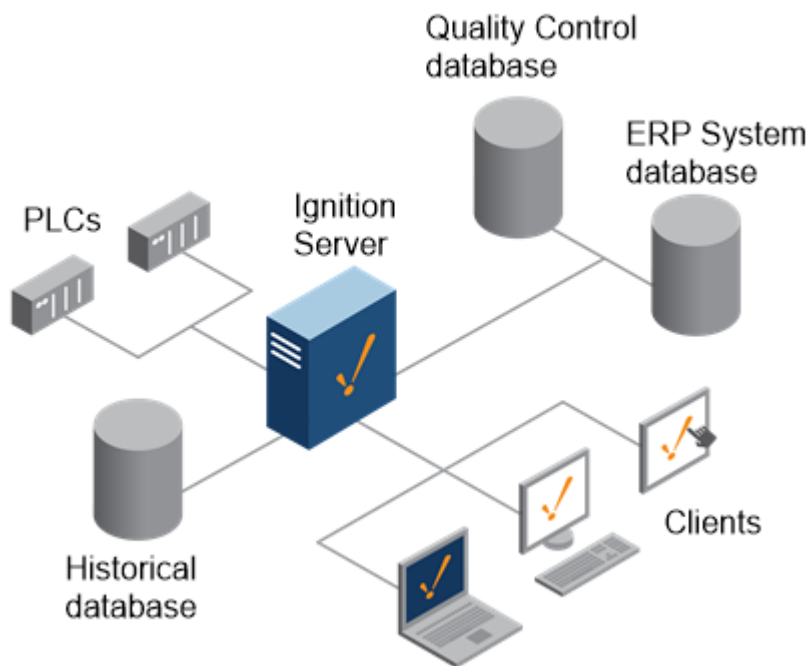
There is a lot of debate on this subject but the short answer to this question is 'whichever database your IT department already uses.' Modern relational database all have the same basic functionality, but slightly different ways of doing things. If your IT department already supports Microsoft SQL Server (MSSQL), then they already have the tools and knowledge to manage those databases. Because they are so similar (and Ignition takes care of so much for you), it is almost always easier to learn to use an existing database than to add IT support for a second type. Not to mention that adding a second type might mean hiring new personnel in the IT department.

If your company does not already have a database preference, then it's up to you to decide which is best based on your needs. They all have a free version, but different limitations. For example: MSSQL has a cap on how much data can be stored in their free version. MySQL does not have a cap like this, but also does not allow phone support for their free version.

Where to Install a Database

You can install your database anywhere that Ignition has access to through the network. There are two main options for installing your database: one is on the computer that Ignition is installed on, and the other is installed on a different server in your network. Technically there is a third option to connect Ignition to a database that is in a remote location using a VPN or some other way to access it. This third option will work, but because of latency and the data being physically very far away, it is not recommended for storing data that will be accessed often like Tag History.

For production systems, we recommend that your database is on it's own server, not installed on the computer with Ignition. This is helpful for many reasons, but mostly because databases can potentially take up a lot of resources on a computer. If the database is on it's own computer, you don't have to worry about other programs starving for memory or CPU. If you do this and install your database on another computer, just make sure to adjust your firewalls and pay attention to the database connection security. Most databases don't allow the default username to connect remotely.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Installing MySQL



INDUCTIVE
UNIVERSITY

Installing MySQL

[Watch the video](#)

To Install the MySQL Server and Workbench

1. Go to the MySQL website at <http://dev.mysql.com/downloads/mysql/>.
2. Scroll-down to **Windows (x86,64-bit), MySQL Installer MSI**.

dev.mysql.com/downloads/mysql/

Please report any bugs or inconsistencies you observe to our [bugs Database](#).
Thank you for your support!

Generally Available (GA) Releases | **Development Releases**

MySQL Community Server 5.6.17

Select Platform:
Microsoft Windows ▼

Recommended Download:

MySQL Installer 5.6 for Windows

All MySQL Products. For All Windows Platforms.
In One Package.

A graphic showing the Windows logo with a stylized 'M' and 'S' inside it, representing the MySQL Installer.

Starting with MySQL 5.6 the MySQL Installer package replaces the server-only MSI packages.

Windows (x86, 64-bit), MySQL Installer MSI [Download](#)

Other Downloads:

Windows (x86, 32-bit), ZIP Archive (mysql-5.6.17-win32.zip)	5.6.17	212.1M	Download
MD5: aef25ffc212fb5edfa25d8a776d181eb6 Signature			
Windows (x86, 64-bit), ZIP Archive (mysql-5.6.17-winx64.zip)	5.6.17	217.1M	Download
MD5: 195c786baa4c61c449a7be21fd10c0d5 Signature			

- NOTE** MySQL Installer is 32-bit, but will install both 32-bit and 64-bit binaries.

Thank you for your support!

5. Click the **Download** button.
6. On the next page, click the **No thanks, just start my download**.

- MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the [Login »](#) link and follow the instructions. If you don't have an Oracle Web account, you can sign up for a free account by clicking the [Sign Up »](#) link and following the instructions.

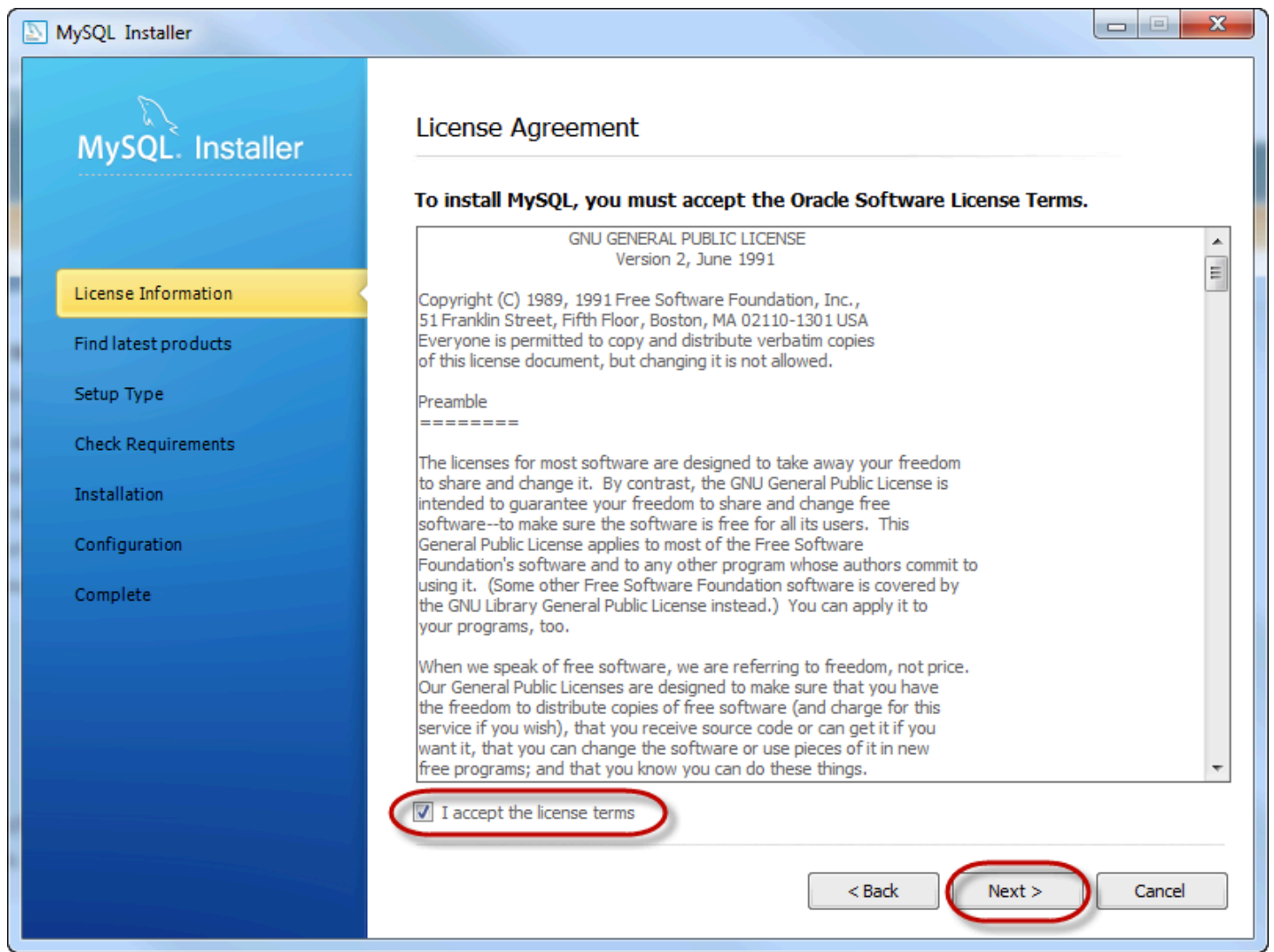
No thanks, just start my download.

The **.msi** file will start downloading.

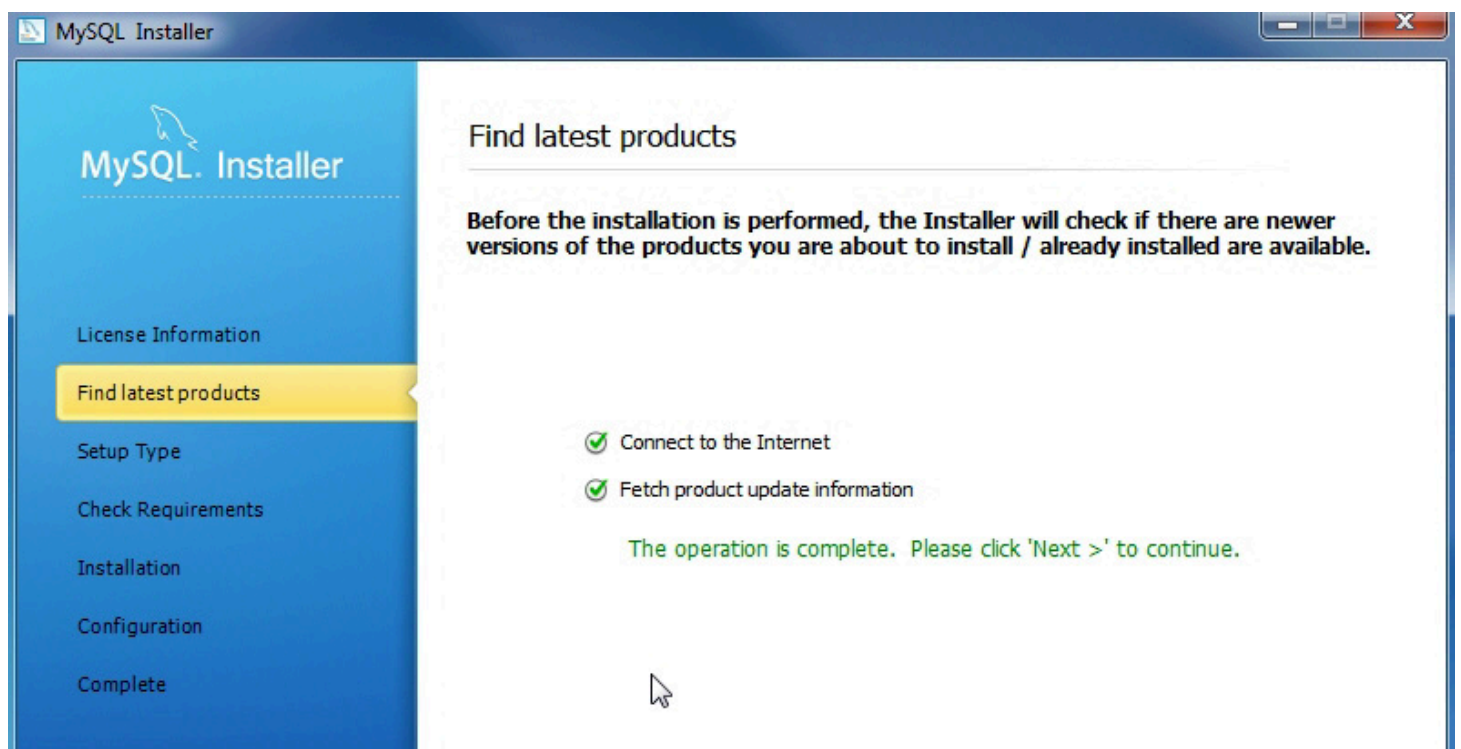
- Click the **.msi** file to run it and to configure the installer.
The **Welcome** window will be displayed.
- Select the **Install MySQL Products** action.



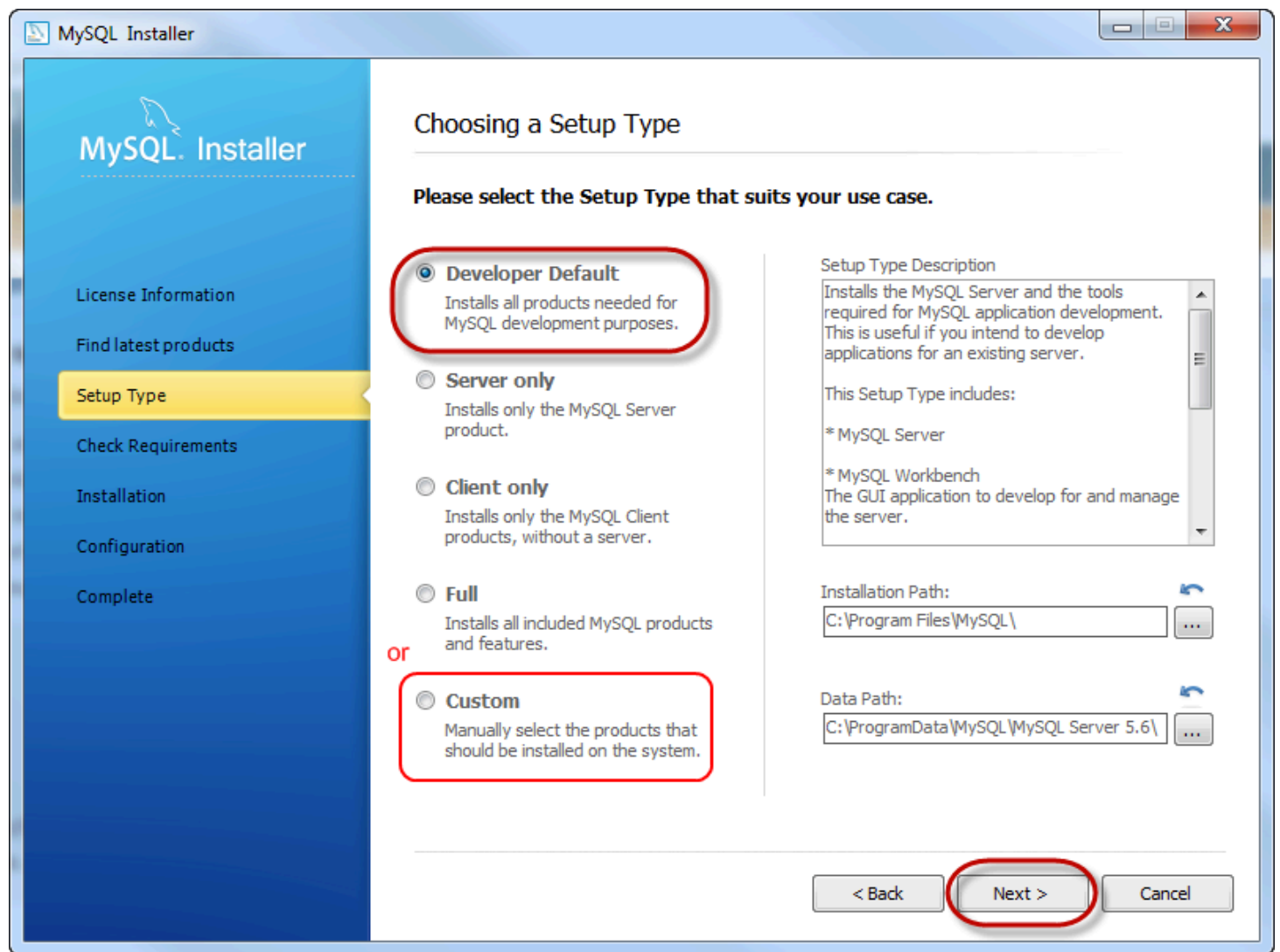
9. On the **License Agreement** page, click the **I accept the license terms** box and click **Next**.



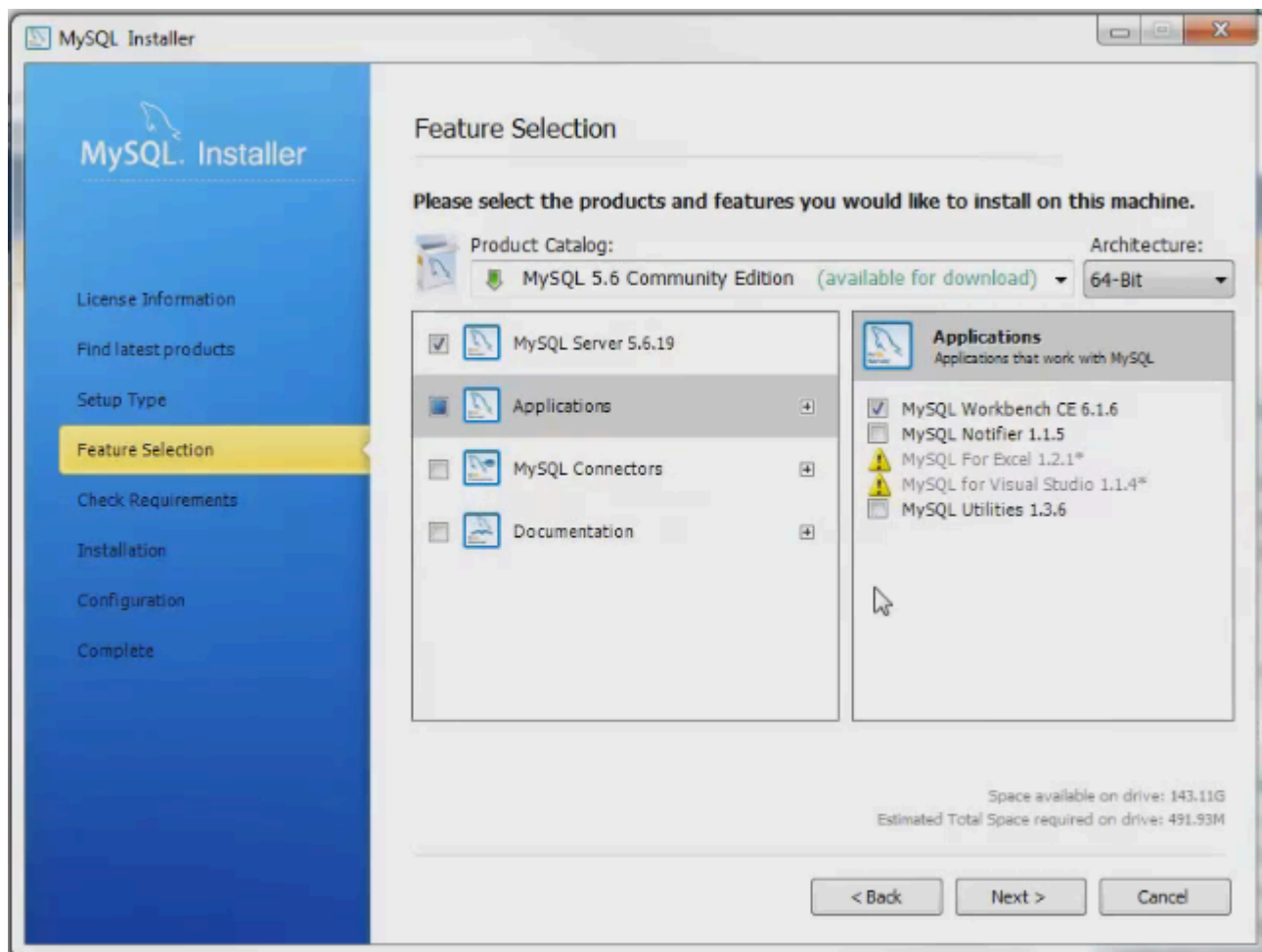
10. On the **Find latest products** page, click **Execute** to check for installer updates from MySQL. If you want to use the version you have, you can select the skip option. When checking is complete, click **Next**.



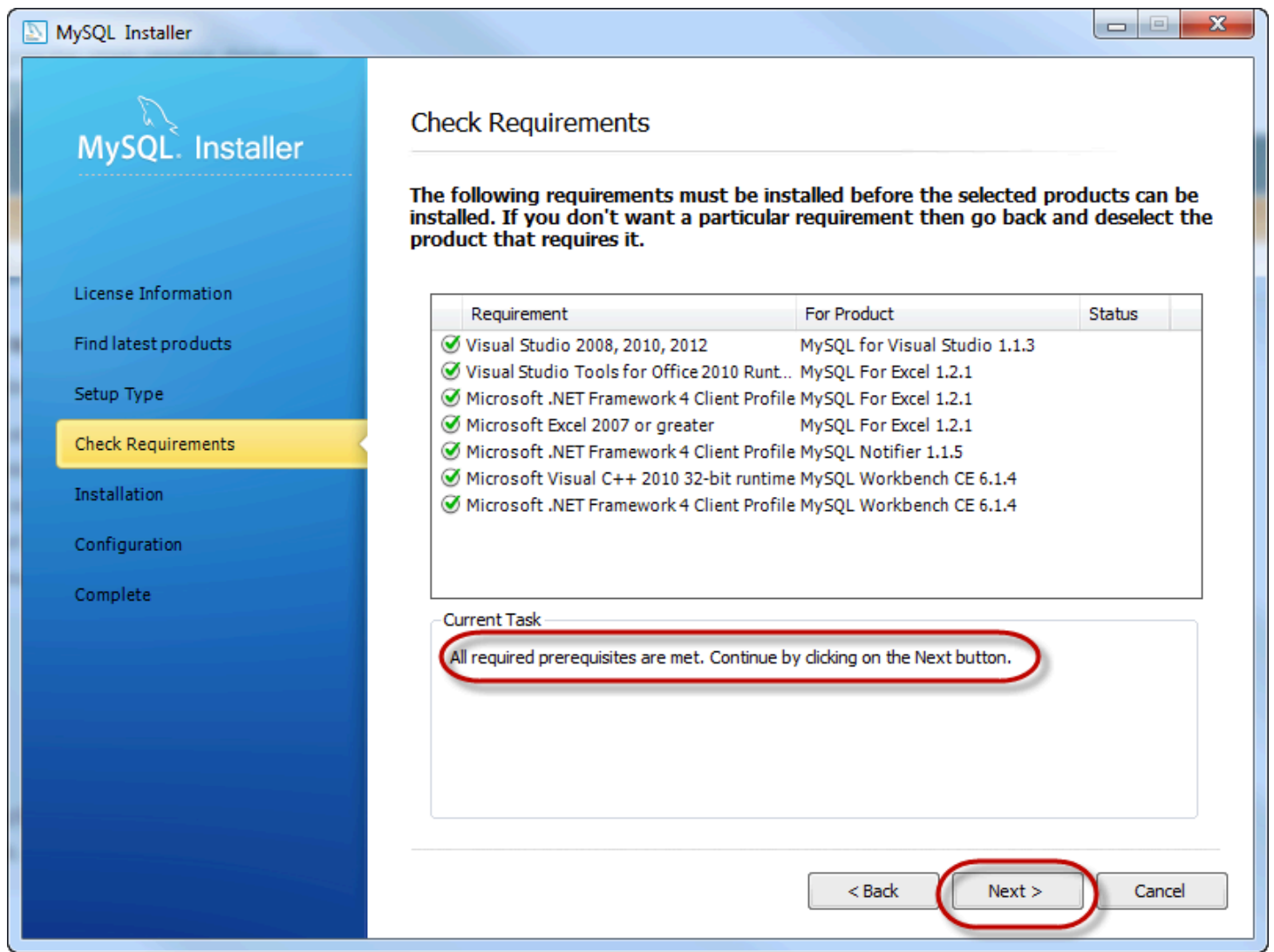
11. On the **Choosing a Setup Type** page, the simplest option is to select **Developer Default** and click **Next**. The only drawback is that the install will be much larger.



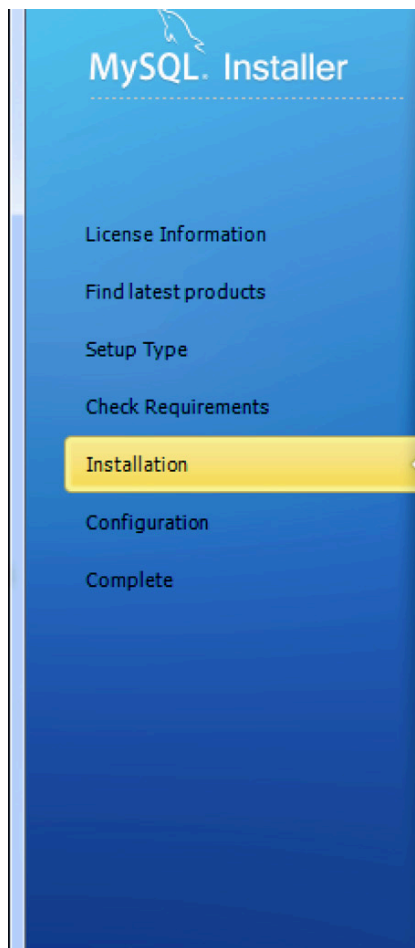
Optionally, you can select the **Custom** option and remove everything except the **MySQL Server** object. You can include the **MySQL Workbench** if you want, it is nice to have if you ever want to modify the raw structures.



12. On the **Check Requirements** page, click **Execute** to make sure all required prerequisites are met. Follow any instructions to install any missing prerequisites. Click **Next** once all prerequisites are met.



13. On the **Installation Progress** page, click **Execute** to install all of MySQL products.
The **MySQL Server** and the **MySQL Workbench** will both be installed. Once all of the products are installed click **Next**.



Installation Progress

The following products will be installed or updated.

	Product	Status	Progress	Notes
✓	MySQL Server 5.6.15	Install success		
✓	MySQL Workbench CE 6.0.8	Install success		
✓	MySQL Notifier 1.1.5	Install success		
✓	MySQL For Excel 1.2.0	Install error		
✓	MySQL Utilities 1.3.6	Install success		
✓	Connector/ODBC 5.2.6	Install success		
✓	Connector/C++ 1.1.3	Install success		
✓	Connector/J 5.1.28	Install success		
✓	Connector/NET 6.8.3	Install success		
✓	MySQL Connector/C 6.1.3	Install success		
✓	MySQL Documentation 5.6.15	Install success		
✓	Samples and Examples 5.6.15	Install success		

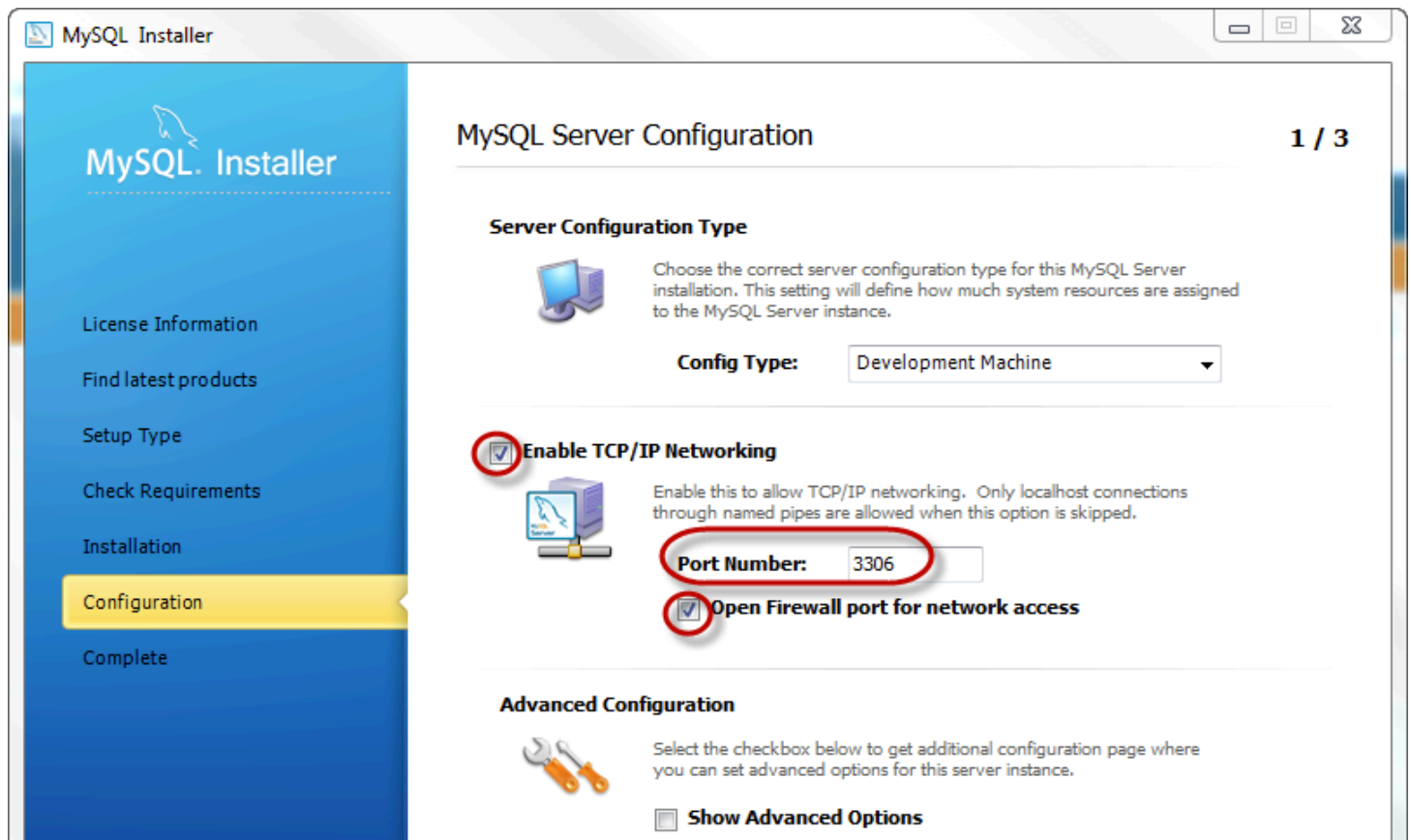
Show Details >

< Back

Next >

Cancel

14. On the **Configuration Overview** page, click **Next** to setup the configuration.
15. On the **MySQL Server Configuration** page (1/3), make sure the default **Port Number** is **3306**, keep all the defaults and click **Next**.



16. On the next **MySQL Server Configuration** page (2/3), choose a password for the root user, type it in, and write it down so you have it for future, then click **Next**.

MySQL Installer

MySQL Server Configuration

2 / 3

Root Account Password

Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password:

Repeat Password:

Password Strength: **Weak**

MySQL User Accounts

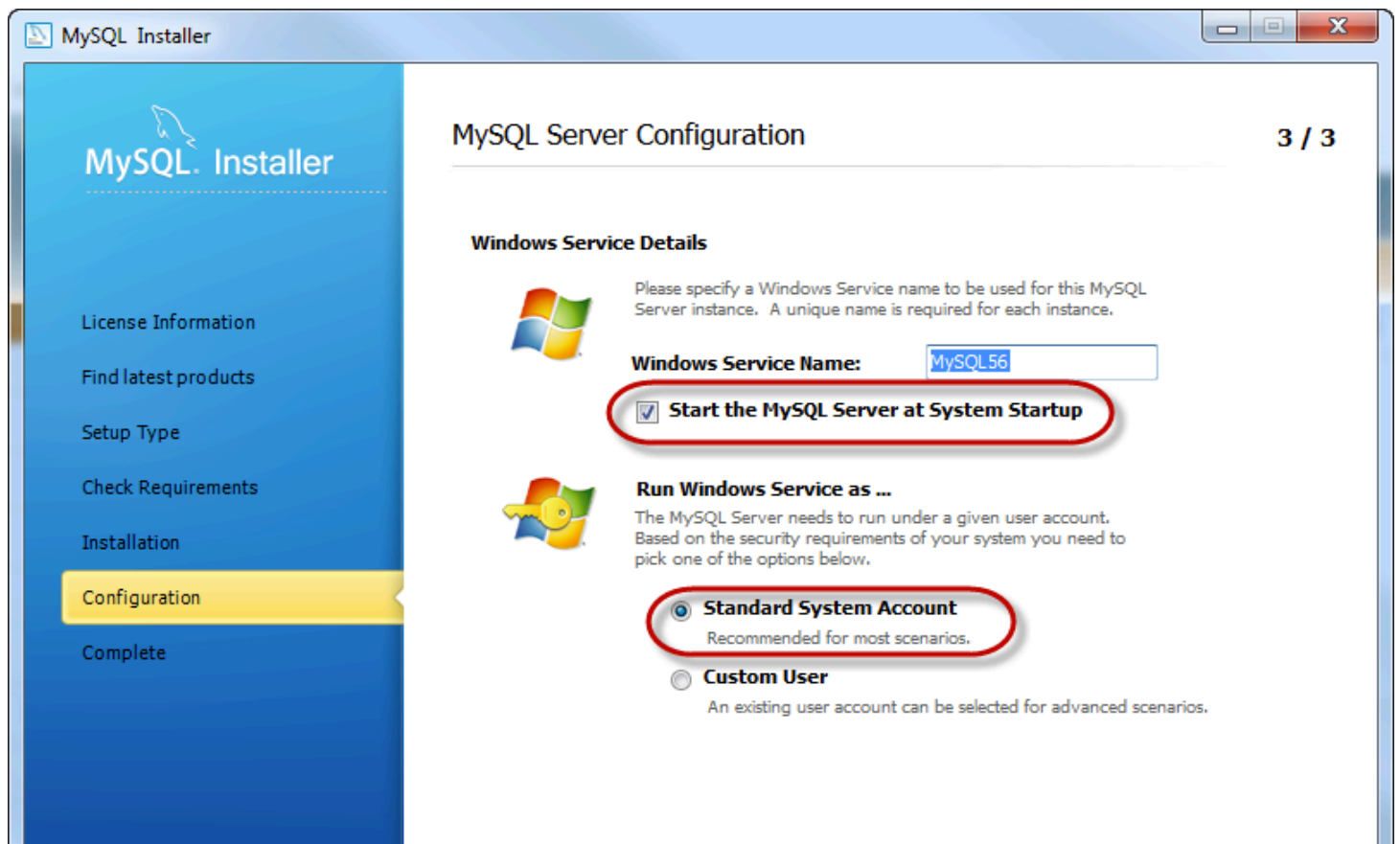
Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL Username	Host	User Role
----------------	------	-----------

Add User
Edit User
Delete User

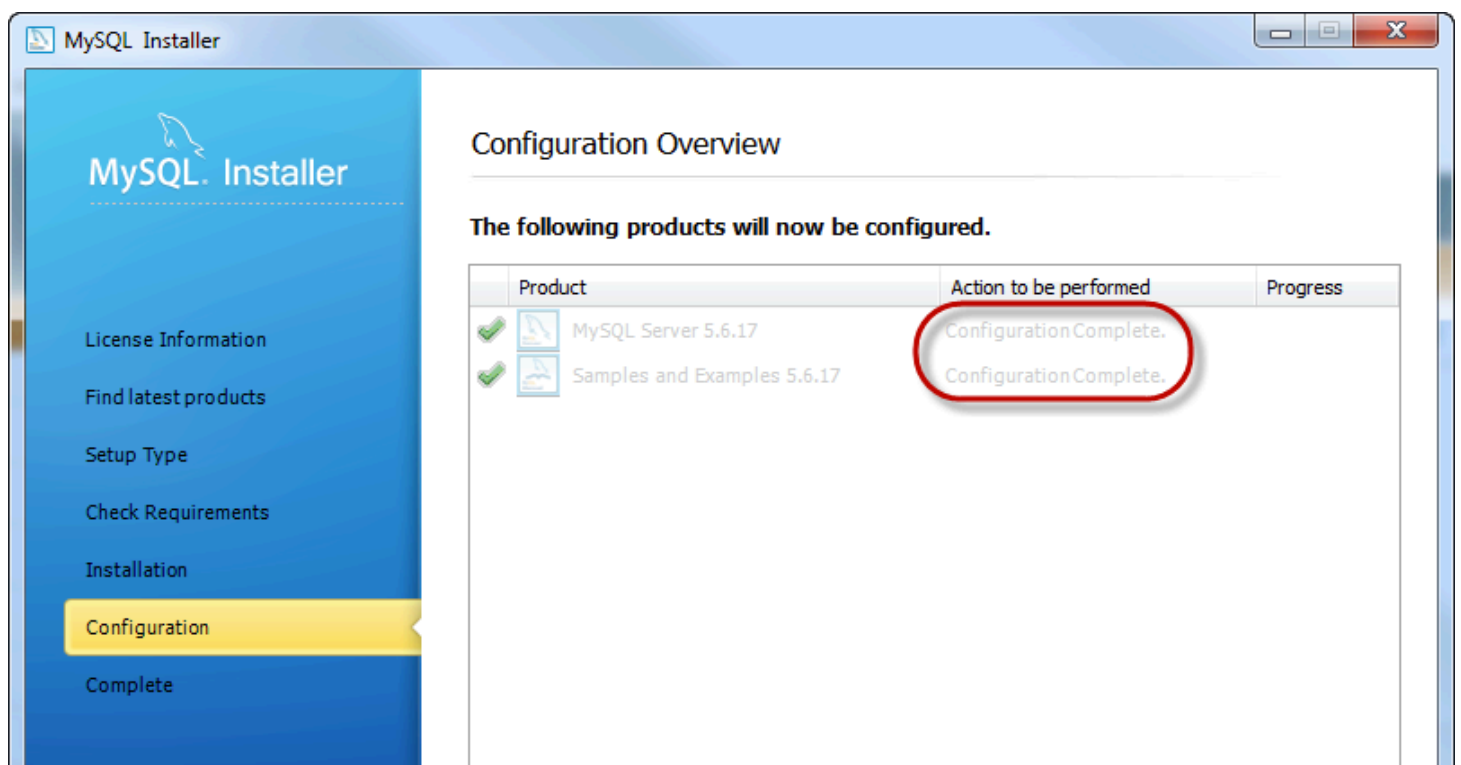
17. Make sure the following two items are selected by default:

- **Start the MySQL Server at System Startup**
- **Standard System Account**

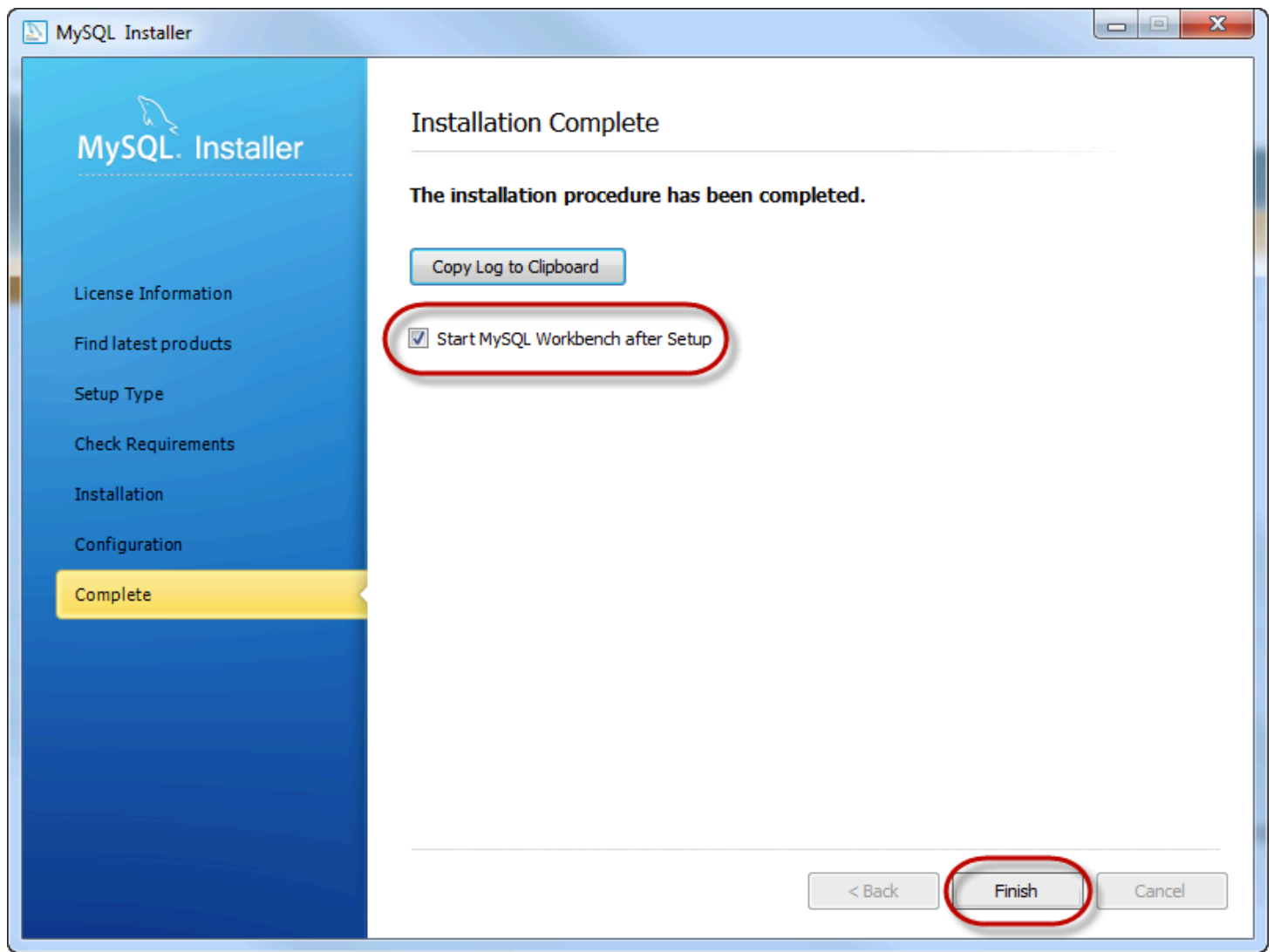


This installs MySQL as a Service and starts MySQL automatically each time you start your computer.

18. Click **Next**. The **Configuration Overview** page is displayed and shows the MySQL server installation is complete.



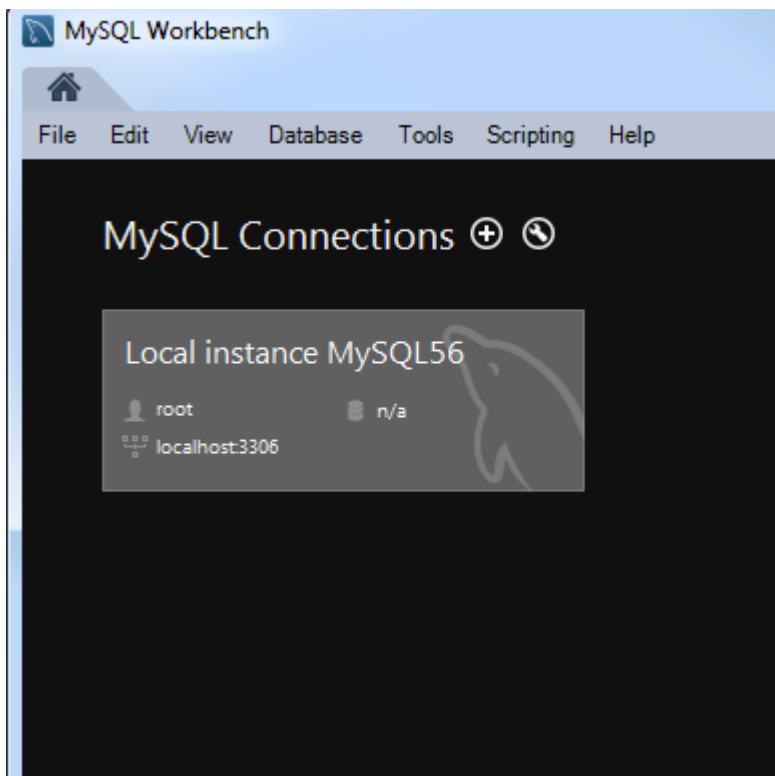
19. Click **Next**.
20. The **Installation Complete** window is displayed and shows the check box for **Start MySQL Workbench after Setup** is selected.



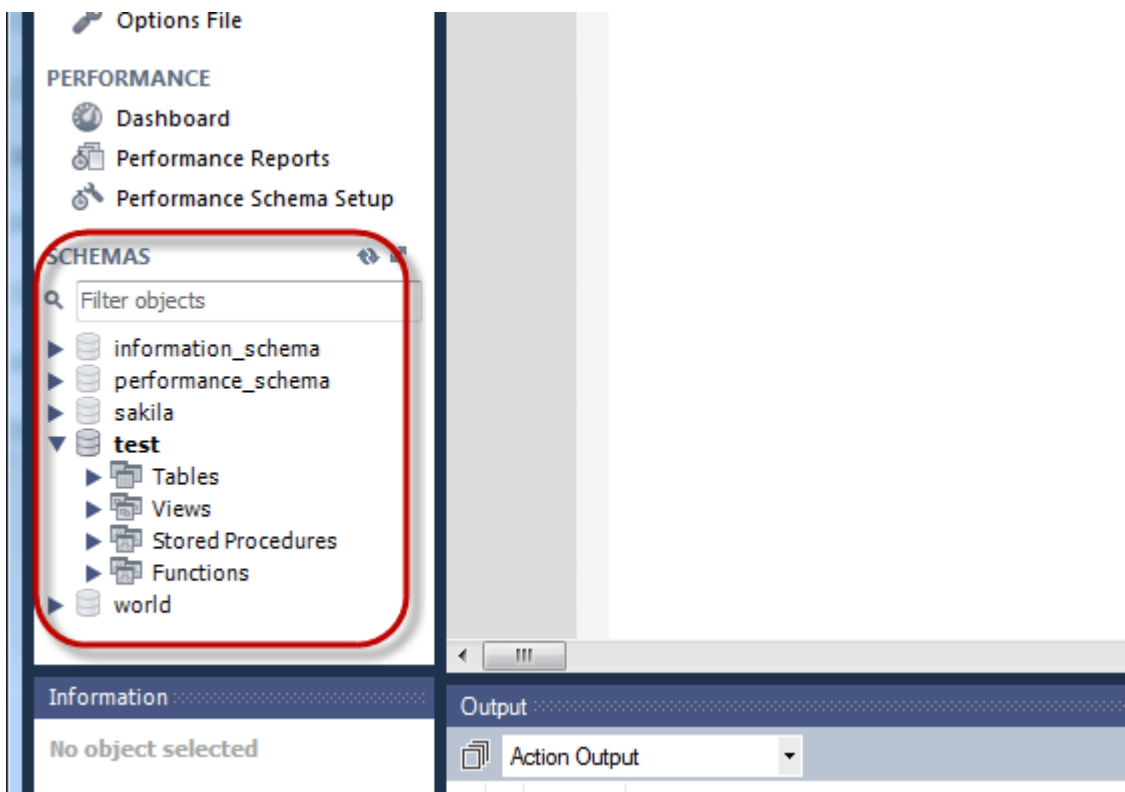
21. Click **Finish** to close the Wizard and complete the installation.

The **MySQL Workbench** window is displayed. MySQL Workbench lets you to administrate the MySQL server.

22. Click on **Local instance MySQL** to connect to the newly installed MySQL server.



23. Enter the root **password** you earlier entered into the installer and click **OK**.
24. The **Local instance MySQL** graphical user interface is displayed.



You are now connected and can see, in the Navigator, the default databases. Older versions of MySQL will start with a database named **test** where new ones may not.

25. If you want to create a new schema with a new name (like **test**), right-click in the white background of the Object Browser (next to the rest of the schemas) and select **Create Schema**. Name it and click the **Apply** button in the lower right.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Installing Microsoft SQL Server Express



INDUCTIVE
UNIVERSITY

Installing Microsoft SQL
Server Express

[Watch the video](#)

You need to download and install both the SQL Server 2014 Express and the SQL Management Studio before you connect to the database.

To Download SQL Server 2014 Express

1. Go to <http://www.microsoft.com/en-us/server-cloud/products/sql-server/>.
2. Click **Products** and under **SQL Server** click on **SQL Server Editions**.
3. Choose the **Express** tab and click on **Download SQL Server 2014 Express**.
4. Click on the **SQLEXPRESS_x64_ENU.exe** file to run the executable.
5. On the **SQL Server Installation Center** window, click on the **New SQL Server stand-alone installation or add features to an existing installation** link.
6. On the **License Terms** window, choose **I accept the license terms** and click **Next**.
7. On the **Product Updates** window, click **Next** to start installing the SQL Server.
8. On the **Feature Selection** window, stay with the default selections and click **Next**.
9. The **Instance Configuration** window shows **Named instance: SQL Express**, you can keep it or change it if you like, click **Next**.
10. On the **Server Configuration** window, choose **Automatic** from the dropdown under **Startup Type** for the **SQL Server Browser** service, and click **Next**.
11. On the **Database Engine Configuration** window, for **Authentication Mode** choose **Mixed Mode**, enter a **password**, and click **Next**.
12. On the next few windows, continue clicking **Next** until it shows installation is **Complete**.
The **SQL Server 2014 Express** is now installed.

Microsoft SQL Server Express installs with a **tempdb** that wipes its data whenever the computer is restarted.

To Download SQL Management Studio

1. Go to <http://www.microsoft.com/en-us/download/details.aspx?id=8961>.
2. Click on the **Download** button.
3. Click on the **SQLManagementStudio_x64_ENU.exe** file to run the executable.
4. On the **SQL Server Installation Center** window, click on the **New SQL Server stand-alone installation or add features to an existing installation** link.
5. On the **Installation Type**, keep the defaults and click **Next**.
6. On the **License Terms** window, choose **I accept the license terms** and click **Next**.
7. On the **Product Updates** window, click **Next** to start installing the SQL Server.
8. On the **Feature Selection** window, stay with the default selection of **Management Tools** and click **Next**.
9. On the **Database Engine Configuration** window, for **Authentication Mode** choose **Mixed Mode**, enter a **password**, and click **Next**.
10. On the next few windows, continue clicking **Next** until it shows installation is **Complete**.
The **Management Tools** is now installed.
11. To run the program, go to **Start > Programs > Microsoft SQL Server > SQL Server Management Studio**.
12. Click on **Connect** to connect to the Microsoft SQL Server.
13. In **Object Explorer**, you can now see some databases under **Databases > System Databases**.
14. Right-click on **Databases** and select **New Databases....**
The **New Databases** window is displayed.
15. In **Database name**, enter **test**, click **Add**, and then **OK**.
Now you can see the **test** database in the **Databases** folder and can connect Ignition to it, see [Connecting to Microsoft SQL Server Express](#).

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Installing Oracle Express



INDUCTIVE
UNIVERSITY

Installing Oracle Express

[Watch the video](#)

To Download and Install Oracle Express

1. Go to the Oracle website at <http://www.oracle.com>.
2. Click on **Downloads**, and then under **Database**, click on **Oracle Database 11g Express Edition**.
3. Accept the license agreement and select the download version you are interested in.
For example, you can download the **Oracle Database 11g Release 2 Express Edition for Windows 64** database.
The **OracleXE112_Win64.zip** file is downloaded.
4. Unzip the **.zip** file by right-clicking on the file and selecting **Extract All...**, then click **Extract** on the next window.
You will now see a DISK1 folder, and inside it is the **setup.exe** executable file.
5. Double-click on **setup.exe** to start the Oracle database installation.
6. On the **Welcome to Install Wizard** window, click **Next**.
7. Accept the terms in the license agreement, and click **Next**.
8. Enter a password, and click **Next**.
This is the password to the SYSTEM account and you can use any password you wish just note it down.
9. Continue on the next windows, click **Install** to install the database, and click **Finish** once installation is complete.
10. If the completion of the previous step does not automatically take you to the database administrative window in your browser, you can manually open your browser and access the database administrative screen by typing the following URL into your browser:
`http://localhost:8080`.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Installing PostgreSQL



INDUCTIVE
UNIVERSITY

Installing PostgreSQL

[Watch the video](#)

To Install the PostgreSQL Database

1. Go to the **Postgre SQL** website at: <http://www.postgresql.org>
2. Click on **Downloads**, look for and click on the **Windows** link.
3. On the **Windows installers** page, find the **Download** link and click on it, and on the next page select the installer you are interested in.
For example, you can select the **Wins x86-32**.
4. From your Download folder on your computer, click on the Postgre **.exe** file to install the database.
5. Go through all the windows of the **Install Wizard** until installation is complete.
6. From the Windows **Start** menu, open the **Postgre pgAdmin** database.
7. In the **Object browser** of the **pgAdmin** window, right-click on **PostgreSQL**, select **Connect**, enter your **password** to connect to the Server, and click **OK**.
You will now see the **Databases** folder in the Object browser.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Connecting to Databases

Connect Once

Many of the advanced features of Ignition, such as the Transaction Groups and Tags Historian require a connection to an external database and most databases require special permissions for each computer that wants to connect. Fortunately Ignition takes care of all of this for us. You can create a connection to your database once and every system in Ignition will use that central connection. There's no need to worry about updating your database settings to add another client.

This central database connection also makes it easy to swap between databases or schemas. You can tell every query to use the default connection, then just change the default to update everything. Alternatively, you can force specific queries or systems to use a particular connection. Create as many database connections as you want and start designing using all of them.

Common Connection Settings

Now that we've installed your database, let's connect to it. You can find detailed descriptions for many database connections in this User Manual, however they all include the following three main steps:

1. **Add a Database Connection:** Once you are in the Gateway **Configure** section of the Gateway's web interface, use the menu on the left to go to the **Databases > Connections** section. On at the **Database Connections** page, click on the **Create new Database Connection...** link at the bottom of the table.
2. **Pick a JDBC Driver:** Ignition connects to databases using JDBC drivers that are unique to each database. Drivers for the most popular databases are included so there is usually no need to install the JDBC driver manually. Ignition ships with drivers for Microsoft SQL Server, MySQL, Oracle, and PostgreSQL. Pick the JDBC driver for your database, and click on the **Next** button. If a suitable driver is not available in the list, you need to add a new JDBC driver for other databases, like IBM DB2, which is not very difficult to do, see Adding a JDBC Driver.
3. **Configure the Connection:** After selecting the driver, you'll configure the settings for the connection. Some settings, such as the Connect URL are specific to the driver that you're using.

Main Databse Connection Properties

Property	Description
Name	Each database connection needs a unique name, which consists of letters, numbers and underscores.
Description	A brief description of the database.
JDBC Driver	The JDBC driver dictates the type of database that this connection can connect to. It cannot be changed once created.
Connect URL	A string that instructs the driver how to connect to the database. This string is the server address, and may include the port, instance name, database name, and so on. The format and parameters depend on the driver being used.
Username	The username to use when connecting. Some databases support other authentication methods, such as Windows authentication, in which case this field is not used.
Change Password?	Check the box to change the existing password.
Password	Enter password
Password	Re-type password for verification.
Extra Connection Properties	Depending on which database you are connecting to, there will be different default values placed in this box. MS SQL Server requires you to place your database name here, but for other databases you can usually leave this at its default values. Each database has its own set of available extra connection properties so you must refer to your Database documentation to determine what is valid here.
Enabled	Lets you to enable or disable a database connection.
Validation Timeout	The time in milliseconds between database validation checks. (default: 10,000)

Property	Description
Failover Datasource	The connection that is automatically used when this connection is not available.
Failover Mode	Lets you select how to handle the database connection failing and recovering. Database connections support failover, this means that the objects which use a database connection, will use a different connection if the one they are using becomes unavailable. The Failover Datasource property determines which connection is used, and the Failover Mode determines when, if ever, the connection is switch back to the primary connection. There are two failover modes: - STANDARD mode means that this datasource will fail over when a connection cannot be retrieved, but when connectivity is restored, connections will again come from this datasource. - STICKY mode means that once this datasource fails over, connections will continue coming from the failover datasource until the failover datasource itself fails or the Gateway is restarted.
Slow Query Log Threshold	Queries that take longer than this amount of time, in milliseconds, will be logged. This helps to find queries that are not performing well. (default: 60,000)

Advanced Settings

There are a many advanced settings that you don't need to change under normal circumstances. See the description for each property on the settings page.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Connecting to MySQL



INDUCTIVE
UNIVERSITY

Connecting to MySQL

[Watch the video](#)

Overview

This page details how to configure a connection to a MySQL database. Note that in some cases you may need to update your JDBC driver, so it is a good idea to consult the table of [supported databases](#).

To Connect Ignition to MySQL Database

1. From the Gateway, go to the **Configure** section.
2. From the menu on the left, select **Databases > Connections**.
The Database Connections page will be displayed.
3. Look for the blue arrow, click on **Create new Database Connection...**

The screenshot shows the Ignition Gateway web interface. At the top, there's a dark blue header with the Ignition logo and navigation links like 'HOME', 'STATUS', and 'CONFIGURE'. A green banner across the top indicates 'License Incomplete' with a timer at 0:00:00 and buttons for 'Reset Trial' and 'View Modules'. On the left, a dark blue sidebar contains a search bar and a menu with categories like 'SYSTEM', 'NETWORKING', and 'SECURITY'. Under 'SYSTEM', 'Connections' is selected. The main content area is titled 'Database Connections' and contains a table with columns: Name, Description, JDBC Driver, Translator, and Status. The table is currently empty, showing 'No Database Connections'. Below the table, there is a blue button with a right-pointing arrow and the text 'Create new Database Connection...'. A light blue note box below the button states: 'Note: For details about a connection's status, see the Database Connection Status page.'

4. Select the **MySQL ConnectorJ** driver and click **Next**.

Add Connection Step 1: Choose Driver

 Select the correct JDBC Driver for the type of database you wish to connect to. If no driver corresponds to your database, go to the Driver Configuration page to add a new driver.

☐ **Firebird JDBC Driver**

The Jaybird JDBC driver for Firebird

☐ **IBM DB2**

The official IBM DB2 JDBC Driver.

☐ **Microsoft SQLServer JDBC Driver**

The Microsoft SQL Server JDBC Driver is a Java Database Connectivity (JDBC) 4.0 compliant driver.

☒ **MySQL ConnectorJ**

The official MySQL JDBC Driver, Connector/J.

☐ **Oracle JDBC Driver**

The Oracle Database JDBC driver.

☐ **PostgreSQL JDBC Driver**

The official PostgreSQL JDBC Driver.

Next >

NOTE

Database connections in Ignition are powered by JDBC drivers. Ignition ships with drivers for Microsoft SQL Server, MySQL, and a few others.

5. On the **New Database Connection** page, enter the following information:

- Name: **MySQL** (Use a meaningful name such as **MySQL**)
- Connect URL: **jdbc:mysql://localhost:3306/test** (By default, MySQL creates an empty database called **test**)

Controller | Master

USER MANUAL
SUPPORT

admin
Sign Out

Launch Designer

Ignition
by Inductive Automation

HOME
STATUS
CONFIGURE

Search...

License Incomplete ? 0:00:00
Reset Trial
View Modules

SYSTEM

Overview
Backup/Restore
Licensing
Modules
Projects
Redundancy
Gateway Settings

NETWORKING

Gateway Network
Email Settings

SECURITY

Auditing
Users, Roles
Service Security
Security Zones

DATABASES

Connections
Drivers
Store and Forward

ALARMING

General
Journal
Notification
On-Call Rosters

New Database Connection

Main Properties

Name

MySQL

Choose a name for this database connection.

Description

JDBC Driver

MySQL Connector/J

The JDBC driver dictates the type of database that this connection can connect to. It cannot be changed once created.

Connect URL

jdbc:mysql://localhost:3306/test

The Connect URL is JDBC-driver specific. It usually contains the address of the machine that the database is running on.
The format of the MySQL connect URL is:
jdbc:mysql://host:port/database
With the three parameters (in bold)
host: The host name or IP address of the database server.
port: The port that the database server is running on. MySQL default port is **3306**.
database: The name of the logical database that you are connecting to on the MySQL server.

NOTE

MySQL uses the following **Connect URL** format: **jdbc:mysql://hostaddress:3306/database**

Where **hostaddress** is the address of the machine with MySQL installed, for example: localhost, 192.168.1.1, db-server, etc. and **database** specifies the database the database schema the connection will target. It's important to understand that a MySQL server can host many database files. The connection will target one database.

6. To configure the connection, enter the following information:

- Username: **root**
- Password: **mysql** (Password is what you entered during MySQL server installation, **mysql** is the password for this example)
- Extra Connection Properties: Leave at the default value. By default, the **zeroDateTimeBehavior** parameter is defined and it is usually not necessary to add more parameters unless you are planning to store text outside of the range of 7-bit ASCII (a-z). If you are, add **useUnicode=yes;characterEncoding=utf8;** to the end of the extra connection properties, including the semi-colons.

Connect URL	jdbc:mysql://localhost:3306/test The Connect URL is JDBC-driver specific. It usually contains the address of the machine that the database is running on. The format of the MySQL connect URL is: jdbc:mysql://host:port/database With the three parameters (in bold) host : The host name or IP address of the database server. port : The port that the database server is running on. MySQL default port is 3306 . database : The name of the logical database that you are connecting to on the MySQL server.	
Username	root	
Change Password?	☐ Check this box to change the existing password.	
Password	*****	
Password	***** Re-type password for verification.	
Extra Connection Properties	zeroDateTimeBehavior=convertToNull;connectTimeout=120000;socketTimeout=120000; There is an extensive list of extra connection properties available for MySQL Connector/J. See the documentation for a table describing all connection properties.	
Enabled	☒ Disabling a connection will prevent communication to the target database. (default: true)	
Validation Timeout	10000 The time in milliseconds between database validation checks. (default: 10,000)	
Failover Datasource	- none - Another datasource that will be used to handle queries if this datasource faults.	
Failover Mode	STANDARD STANDARD Standard failover mode means that this datasource will fail over when a connection cannot be retrieved, but when connectivity is restored, connections will again come from this datasource. STICKY Sticky failover mode means that once this datasource fails over, connections will continue coming from the failover datasource until the failover datasource itself fails or the Gateway is restarted. (default: STANDARD)	
Slow Query Log Threshold	60000 Queries that take longer than this amount of time, in milliseconds, will be logged. This helps to find queries that are not performing well. (default: 60,000)	

☐ Show advanced properties

7. Click on **Create New Database Connection** at the bottom of the form. Your connection is now created and the Database Connections page is displayed showing the **Status** of your connection as **Valid**.

Database Connections

Name	Description	JDBC Driver	Translator	Status	
MySQL		MySQL ConnectorJ	MYSQL	Valid	<button>delete</button> <button>edit</button>

→ [Create new Database Connection...](#)

Note: For details about a connection's status, see the [Database Connection Status](#) page.

8. To display the details about the status of your database connection, click the **Database Connection Status** link in the **Note** under the Database Connections list. This will display the connection status or any errors if the status is Faulted.

Controller | Master

USER MANUAL | SUPPORT

admin | Sign Out

Launch Designer

Ignition
by inductive automation

HOME STATUS CONFIGURE

Search...

SYSTEMS

Overview

Performance

Alarm Pipelines

Gateway Scripts

Modules

Redundancy

Reports

SFCs

Voice Alarming

Tags

EAM Tasks

Transaction Groups

CONNECTIONS

EAM Agents

Databases

License Incomplete ? 0:00:00

Reset Trial View Modules

Connections

Databases

Configuration

Valid Connections

1 / 1

Total Throughput

11.7 queries/sec

Name	Driver	Status	Connections	Throughput	
MySQL	MySQL ConnectorJ	Valid	0 / 8	11.7 queries/sec	Details

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Connecting to Microsoft SQL Server Express



INDUCTIVE
UNIVERSITY

Connecting to Microsoft SQL
Server Express

[Watch the video](#)

Overview

This page details how to configure a connection to a SQL Server database. Note that in some cases you may need to update your JDBC driver, so it is a good idea to consult the table of [supported databases](#).

Connecting to Microsoft SQL Server

1. From the Gateway, go to the **Configure** section.
2. From the menu on the left, select **Databases > Connections**.
The **Database Connections** page will be displayed.
3. Look for the orange arrow, click on **Create new Database Connection...**
4. Select **Microsoft SQLServer JDBC Driver** and click **Next**.

Controller | Master

USER MANUAL SUPPORT

admin Sign Out

Launch Designer

Ignition
by inductive automation

HOME STATUS CONFIGURE

Search...

License Incomplete ? 0:00:00 Reset Trial View Modules

SYSTEM

- Overview
- Backup/Restore
- Licensing
- Modules
- Projects
- Redundancy
- Gateway Settings

NETWORKING

- Gateway Network
- Email Settings

SECURITY

- Auditing
- Users, Roles
- Service Security
- Security Zones

DATABASES

- Connections
- Drivers
- Store and Forward

ALARMING

- General
- Journal
- Notification
- On-Call Rosters
- Schedules

TAGS

Add Connection Step 1: Choose Driver

Select the correct JDBC Driver for the type of database you wish to connect to. If no driver corresponds to your database, go to the Driver Configuration page to add a new driver.

- ☐ Firebird JDBC Driver
The Jaybird JDBC driver for Firebird
- ☐ IBM DB2
The official IBM DB2 JDBC Driver.
- ☒ Microsoft SQLServer JDBC Driver
The Microsoft SQL Server JDBC Driver is a Java Database Connectivity (JDBC) 4.0 compliant driver.
- ☐ MySQL ConnectorJ
The official MySQL JDBC Driver, Connector/J.
- ☐ Oracle JDBC Driver
The Oracle Database JDBC driver.
- ☐ PostgreSQL JDBC Driver
The official PostgreSQL JDBC Driver.

Next >

The **New Database Connection** page will be displayed.

5. On the **New Database Connection** page, enter the following information:

Name: **SQLServer**

Connect URL: `jdbc:sqlserver://localhost\SQLEXPRESS`

username: **sa**

password: **sqlserver** (password is what you entered during the SQL Server installation. For this example, password is **sqlserver**)

(i) NOTE

We are connecting to the express edition of SQL Server using the default instance name in the **Connect URL** field. If you have the full SQL Server with default settings, replace SQLEXPRESS with MSSQLSERVER.

Controller | Master

USER MANUAL | SUPPORT

admin | Sign Out

Launch Designer

Ignition
by inductive automation

HOMESTATUSCONFIGURE

Search...

License Incomplete ? 0:00:00

Reset Trial

View Modules

SYSTEM

Overview

Backup/Restore

Licensing

Modules

Projects

Redundancy

Gateway Settings

NETWORKING

Gateway Network

Email Settings

SECURITY

Auditing

Users, Roles

Service Security

Security Zones

DATABASES

Connections

Drivers

Store and Forward

ALARMING

General

Journal

Notification

On-Call Rosters

Schedules

TAGS

History

Realtime

OPC-UA SERVER

Certificates

Devices

Settings

OPC CONNECTIONS

Servers

Quick Client

MOBILE

Settings

New Database Connection

Main Properties

Name

SQLServer

Choose a name for this database connection.

Description

JDBC Driver

Microsoft SQLServer JDBC Driver

The JDBC driver dictates the type of database that this connection can connect to. It cannot be changed once created.

Connect URL

jdbc:sqlserver://localhost\SQLEXPRESS

The Connect URL is JDBC-driver specific. It usually contains the address of the machine that the database is running on.
The format of the SQL Server connect URL is:
jdbc:sqlserver://**host**\instanceName[:port]
With the three parameters (in bold)
host: The host name or IP address of the database server.
instanceName: (optional) the instance to connect to on the host. If not specified, a connection to the default instance is made.
port: (optional) the port to connect to. The default is 1433. If you are using the default, you can omit the port and the preceding ':\'.

For SQL Server, you specify the database name to connect to using the `databaseName` property in the *Extra Connection Properties*.

Username

sa

Password

Password

Re-type password for verification.

Extra Connection Properties

databaseName=test

Use `databaseName=YOUR_DATABASE` to specify the database to connect to.

6. At the bottom of the form, click on **Create New Database Connection**.

Your connection is now created and the **Database Connections** page is displayed showing the **Status** of your connection as **Valid**.

Database Connections

✓ Successfully created new Database Connection "SQLServer"

Name	Description	JDBC Driver	Translator	Status	
MySQL		MySQL ConnectorJ	MYSQL	Valid	delete edit
SQLServer		Microsoft SQLServer JDBC Driver	MSSQL	Valid	delete edit

→ [Create new Database Connection...](#)

Note: For details about a connection's status, see the [Database Connection Status](#) page.

7. To display the details about the status of your database connection, see the **Note** on the above window and click on the **Database Connection Status** link.

Connections

Configuration

Databases

Valid Connections

1 / 2

Total Throughput

12.0^{queries/sec}

Name	Driver	Status	Connections	Throughput	
SQLServer	Microsoft SQLServer JDBC Driver	✓ Valid	0 / 8	12.0 queries/sec	Details

Microsoft SQL Server Connection Guide

This guide helps you with any difficulties you may have in getting the correct settings and parameters when connecting Ignition to Microsoft SQL Server, a popular and robust relational database.

Multiple Instances of Database

Microsoft SQL Server supports multiple instances of the database running concurrently on the same computer. Each instance has its own name and set of system and user databases that are not shared between instances.

Applications, such as Ignition, can connect to each instance on a computer in much the same way they connect to databases running on different computers.

By default, each instance gets assigned a dynamic TCP/IP port on startup that listens for any incoming requests. Since the port is dynamic and the application does not know what the new port is, it must connect using the instance name.

So if the communication is over TCP/IP and the application knows the instance name, how does the application find which port to communicate to? The answer is the Microsoft SQL Server Browser service. The **Microsoft SQL Server Browser** program runs as a Windows service and listens for all incoming requests for resources and provides information, such as the TCP/IP port, about each instance installed on the computer. Microsoft SQL Server Browser also contributes to these two actions: browsing a list of available servers and connecting to the correct server instance.

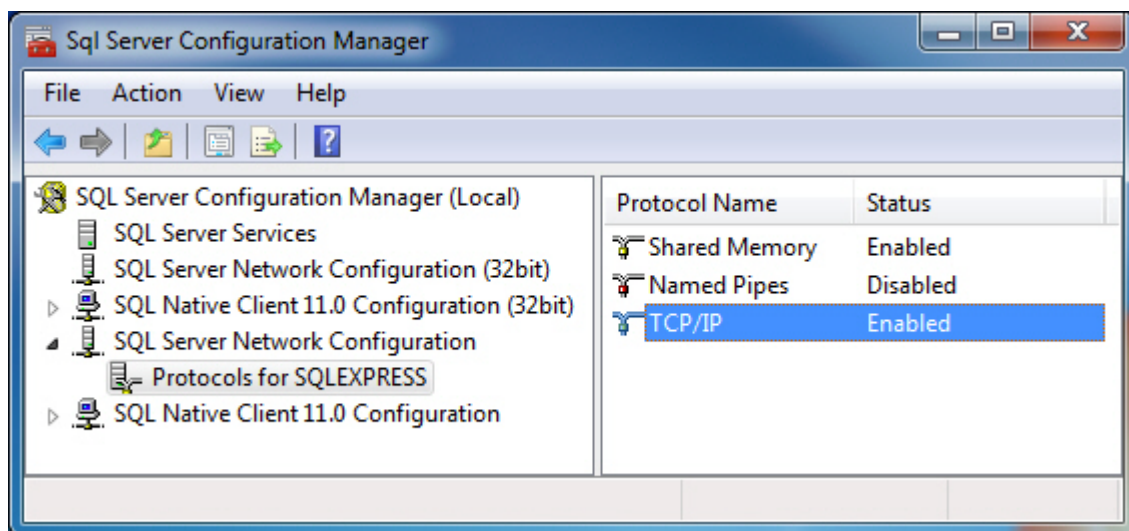
If the Microsoft SQL Server Browser service is not running, you can still connect to SQL Server if you provide the correct port number. For example, you can connect to the default instance of SQL Server with TCP/IP if it is running on port 1433.

Check 1: Make Sure the Database has TCP/IP is Enabled

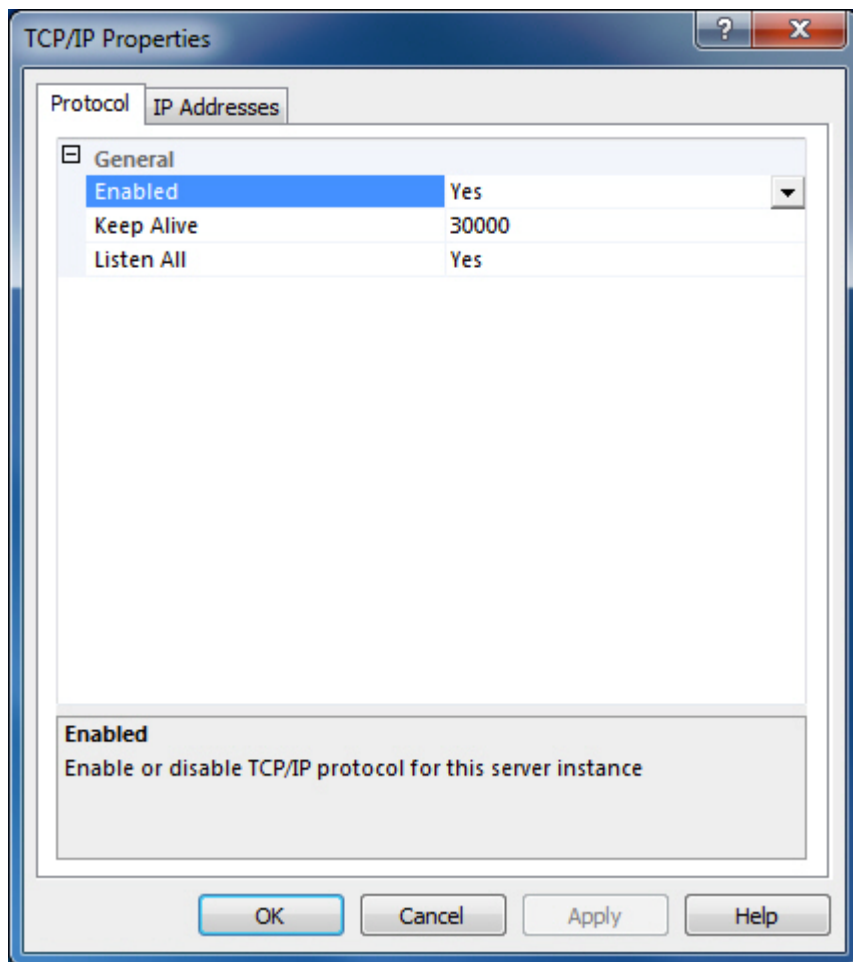
Ignition connects using TCP/IP, therefore make sure your database has TCP/IP enabled.

To check if TCP/IP is enabled

1. Open the **SQL Server Configuration Manager** from **Start > All Programs > Microsoft SQL Server Version # > Configuration Tools > SQL Server Configuration Manager**
The **SQL Server Configuration Manager** window will be displayed.
2. To see all the instances setup on that machine, expand **SQL Server Network Configuration**.
3. Find the database (or instance) you plan on using. To the right, all of the protocols the database supports are shown. Find the **TCP/IP** protocol and select it.



4. Make sure the **Status** next to TCP/IP is **Enabled**. If not, double-click **TCP/IP** and choose **Yes** from the drop-down next to Enabled and click **OK**.



Check 2: Make Sure Microsoft SQL Server Browser is Running

If you ARE connecting to your database using a NAMED INSTANCE, you must make sure that the Microsoft SQL Server Browser is running. As mentioned earlier, the Microsoft SQL Server Browser translates the instance name to a TCP/IP port in order for Ignition to connect to it.

To check if Microsoft SQL Server Browser is running

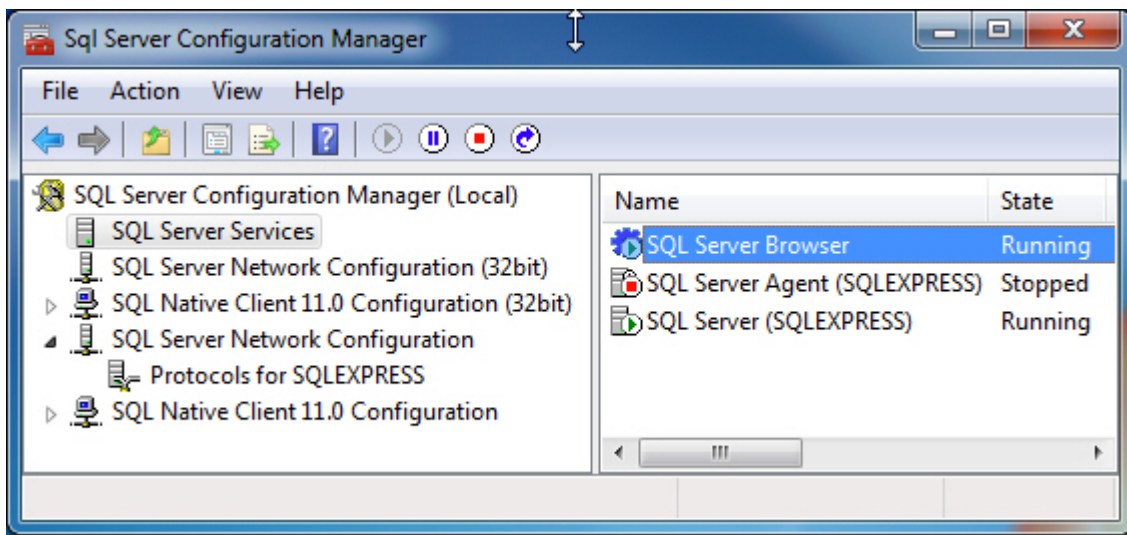
Open the SQL Server Configuration Manager from Start > All Programs > Microsoft SQL Server Version # > Configuration Tools > SQL Server Configuration Manager

Select the SQL Server Services section.

On the right, see all of the services installed. One of the services is SQL Server Browser. Make sure this service is in fact running. If the service is not running, right-click and select Start.

NOTE

The service could be disabled, so you may need to double-click it to enable the service before starting it up.



Different Ways of Connecting to SQL Server

Now that you have ensured that TCP/IP is enabled and the Microsoft SQL Server Browser is running, you can connect to Microsoft SQL Server in four different ways (all using TCP/IP communication) as follows:

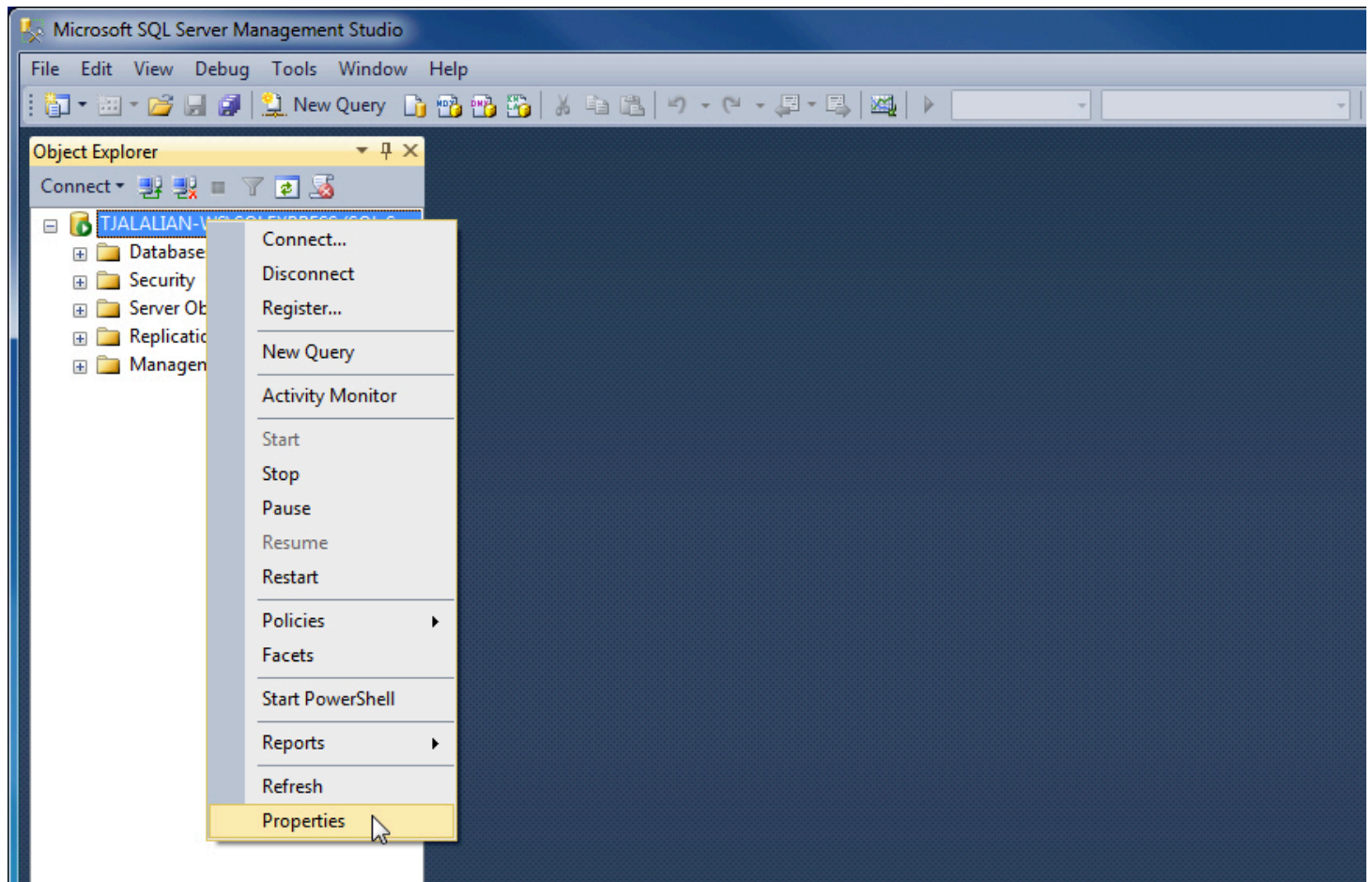
1. Connect using an Instance Name and SQL Authentication
2. Connect using an Instance Name and Windows Authentication (this is the most common method)
3. Connect using a Port and SQL Authentication
4. Connect using a Port and Windows Authentication

Scenario 1: Connect By Using an Instance Name and SQL Authentication

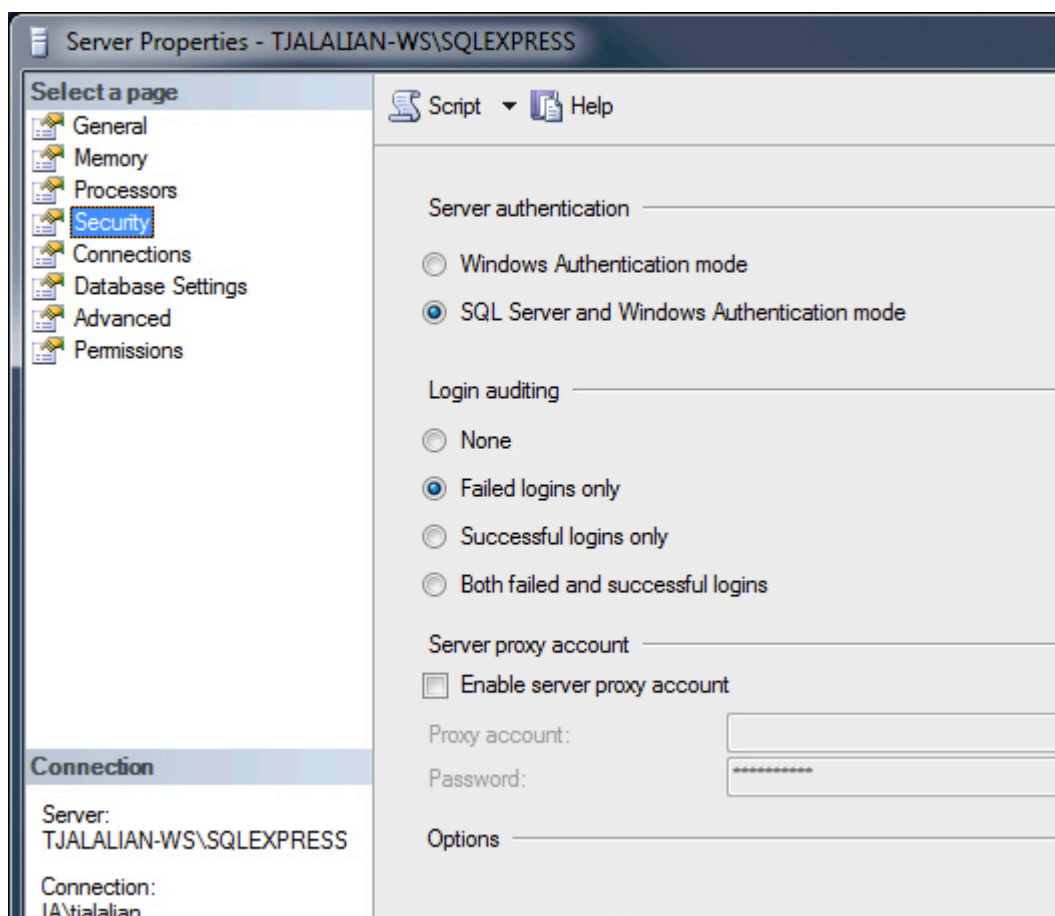
By default, Microsoft SQL Server only allows Windows authentication since it is more secure. But because we are using SQL authentication, we must enable Microsoft SQL to allow this type of authentication.

To enable SQL authentication

1. Open the **Microsoft SQL Server Management Studio** window from **Start > All Programs > Microsoft SQL Server Version # > SQL Server Management Studio**
The window is displayed showing connections to your database.
2. Right-click the top-level database in the **Object Explorer** and select **Properties**.



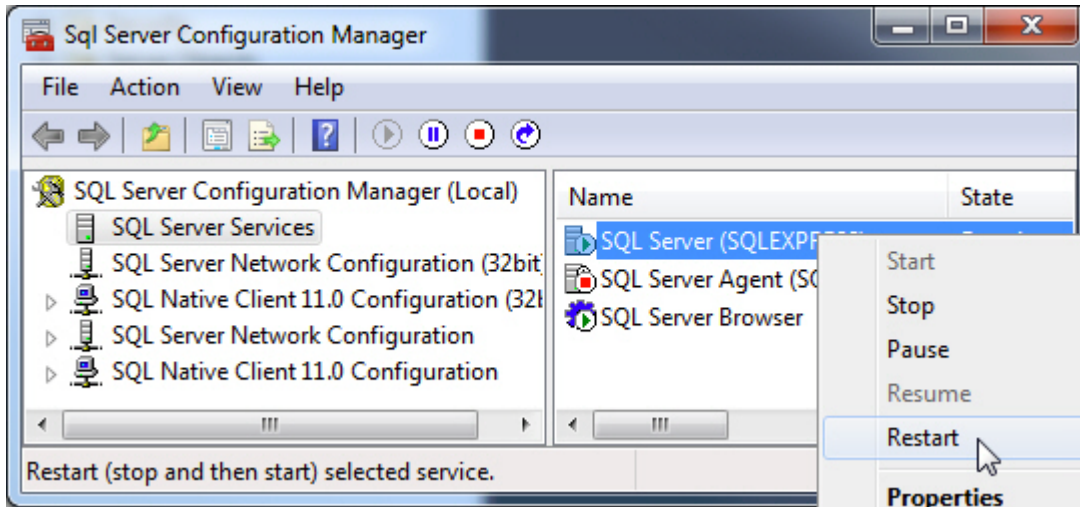
3. From the Server Properties window, on the left side, select **Security**.



4. Verify that **SQL Server and Windows Authentication** mode is selected.
If not, select it and click **OK**.

Now you need to restart the **SQL Server Windows** service so that this setting takes effect.

5. Open the **SQL Server Configuration Manager** at **Start > All Programs > Microsoft SQL Server Version # > Configuration Tools > SQL Server Configuration Manager**.
6. Select the **SQL Server Services** section and restart the **SQL Server (Instance Name)** item.



Now that Microsoft SQL Server accepts SQL authentication, we can configure Ignition.

To Configure the database connection in Ignition

1. Go to and login to the Ignition Gateway configuration page from your webbrowser at `http://hostname:8088/main/web/config/`
2. Select **Databases > Connections** from the menu.
3. Click on **Create new Database Connection**.
4. Select **Microsoft SQL Server JDBC Driver** and click **Next**.
5. In the **New Database Connection** window, enter the following information:

Name: **SQLServer_SQLAuth** (no spaces)

Connect URL: `jdbc:sqlserver://Hostname\InstanceName`

where **Hostname** is your databases IP address or hostname and **InstanceName** is your databases instance name, for example:

`jdbc:sqlserver://localhost\SQLEXPRESS`

`jdbc:sqlserver://10.10.1.5\MSSQLSERVER`

6. Set the username and password to a valid SQL authentication user. For example, **sa** is the default administrator account you can use.
7. To add your own user account, open the SQL Server Management Studio and expand the **Security > Logins** folder.
You will see all the current logins including sa and you can add a new login.
 - To add a new login, right-click on the **Logins** folder and click **New Login...**
The **Login** window is displayed.

- Choose the **SQL Server authentication** mode and type in a Login name and password.

NOTE

You will also have to add permissions to your database by mapping db_datareader and db_datawriter to the new user in the **User Mapping** section of the Login window. If you want Ignition to be able to create tables (ie: for Tag History), you also need to give table creation access such as db_owner.

8. Go back to the **New Database Connection** page in the Gateway, enter the name of your database, for example, in the **Extra Connection Properties** enter: **databaseName=test** (replace **test** with your database name, not the instance name)
9. Click **Create New Database Connection**.
The Database Connection page is displayed showing the Status as **Valid** after a couple of seconds. If the connection is **Faulted**, click on the Database Connection Status link to find out why. Typically, the username/password is incorrect or the user doesn't have the right permissions.

Scenario 2: Connect By Using Instance Name and Windows Authentication

In Windows authentication mode, the username and password used to connect comes from the Ignition Windows Service logon. By default, the Ignition Windows Service is set to local system account which usually doesn't have privileges to connect.

To set up the service to use Windows Authentication

1. Install the **Microsoft JDBC driver package**. The exact version required depends on the version of Java your Gateway is using, and the version of the JDBC Driver Ignition is using must match the downloaded DLL file. If you have any questions regarding which version to download, reach out to our **Support** team for guidance.
2. Extract the files to your desktop. Locate the appropriate DLL from the correct architecture folder (x86 for 32-bit JDBC and x64 for 64-bit JDBC) inside of the `enu/auth` folders in the zip file. Make sure to select the version that matches your database connector, not your computer architecture.
3. Copy the `DLL` file to the **lib** folder in your install directory. If you have the default install directory, it's in the following location: `C:\Program Files\Inductive Automation\Ignition\lib\`
4. The account used to connect will be the account that Ignition is running under in the services menu. To setup Ignition to logon using the right Windows account, open the **Services Control Panel** from **Start > Control Panel > Administrative Tools > Services**
5. Right-click the **Ignition** service and choose **Properties**.
6. Select the **Log On** tab.

7. Choose the **This account** radio button and enter in your Windows username and password.
8. Click **OK** to save.
9. Now restart the Ignition service to make this change take effect. Click the **Action > Restart** button in the menubar to restart the Ignition service (or you can stop and start from the right-click menu).

To configure the database connection in Ignition

1. Go to and login to the Ignition Gateway configuration page from your webbrowser at `http://hostname:8088/main/web/config/`
2. Select **Databases > Connections** from the menu.
3. Click on **Create new Database Connection**.
4. Select **Microsoft SQL Server JDBC Driver** and click **Next**.
5. On the **New Database Connection** page, enter the following information:
 - Name: **SQLServer_WinAuth** (no spaces)
 - Connect URL: `jdbc:sqlserver://Hostname\InstanceName`
 - **Hostname** is your databases IP address or hostname and **InstanceName** is your databases instance name, for example: `jdbc:sqlserver://localhost\SQLEXPRESS`
`jdbc:sqlserver://10.10.1.5\MSSQLSERVER`
 - username: leave blank
 - password: leave blank
 - Extra Connection Properties: **databaseName=test; integratedSecurity=true;** (replace test with your database name)
6. Click on **Create New Database Connection**.

The Status should be Valid after a couple of seconds. Again, if the connection is Faulted, click the Database Connection Status link to find out why.

Scenario 3: Connect By Using Port and SQL Authentication

Connecting by using a port and SQL authentication is just like scenario 1 above except you specify a port instead of the instance name in the **New Database Connection** page.

Enter the following:

Connect URL: `jdbc:sqlserver://Hostname:Port`

where **Hostname** is your databases IP address or hostname and **Port** is your databases TCP/IP port (SQLSERVER default port is 1433), for example: `jdbc:sqlserver://localhost:1433`

`jdbc:sqlserver://10.10.1.5:1433`

Scenario 4: Connect By Using Port and Windows Authentication

Connecting by using a port and Windows authentication is just like scenario 2 above except you specify a port instead of the instance name in the New Database Connection page. Don't forget to download the `sqljdbc_auth.dll` file if you need it.

Enter the following:

Connect URL: `jdbc:sqlserver://Hostname:Port`

where **Hostname** is your databases IP address or hostname and **Port** is your databases TCP/IP port (SQLSERVER default port is 1433), for example: `jdbc:sqlserver://localhost:1433`

`jdbc:sqlserver://10.10.1.5:1433`

Troubleshooting

TCP/IP Communication Not Enabled

SQL Server requires that you explicitly turn on TCP connectivity. To do this, use the SQL Server Configuration Manager, located in the **Start** menu under **Microsoft SQL Server > Configuration Tools**. Under **SQL Server Network Configuration**, select your instance, and then enable TCP/IP in the panel to the right. You need to restart the server for the change to take affect.

Window Firewall

When connecting remotely, make sure that Windows Firewall is disabled, or set up to allow the necessary ports. Normally ports 1434 and 1433 must be open for TCP traffic, but other ports may be required based on configuration.

SQL Server Browser Process Not Running

To connect to a named instance, the **SQL Server Browser** service must be running. It is occasionally disabled by default, so you need to verify that the service is not only running, but set to start automatically on bootup. The service can be found in the Windows Service Manager (**Control Panel > Administrative Tools > Services**).

Mixed Mode Authentication Not Enabled

Unless selected during setup, **mixed mode** or **SQL authentication** is not enabled by default. This mode of authentication is the **username/password** scheme that most users are used to. When not enabled, SQL Server only allows connections using Windows Authentication. Due to the ease of using SQL Authentication over Windows Authentication, we recommend enabling this option and defining a user account for Ignition.

To enable this, open the SQL Server Management Studio and connect to the server. Right click on the instance and select **Properties**. Under **Security**, select **SQL Sever and Windows Authentication mode**.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Connecting to Oracle Express



INDUCTIVE
UNIVERSITY

Connecting to Oracle Express

[Watch the video](#)

Overview

This page details how to configure a connection to an Oracle database. Note that in some cases you may need to update your JDBC driver, so it is a good idea to consult the table of [supported databases](#).

To Connect Ignition to the Oracle Express Database

1. From the Gateway, go to the **Configure** section.
2. From the menu on the left, select **Databases > Connections**.
The **Database Connections** page will be displayed.
3. Look for the blue arrow, click on **Create new Database Connection....**

Database Connections

Name	Description	JDBC Driver	Translator	Status
No Database Connections				

[→ Create new Database Connection...](#)

Note: For details about a connection's status, see the [Database Connection Status](#) page.

4. Select the **Oracle JDBC Driver** and click **Next**.

Controller | Master

USER MANUAL SUPPORT

admin Sign Out

Launch Designer

Ignition
by Inductive automation

HOME STATUS CONFIGURE

Search...

License Incomplete ? 0:00:00 Reset Trial View Modules

SYSTEM

- Overview
- Backup/Restore
- Licensing
- Modules
- Projects
- Redundancy
- Gateway Settings

NETWORKING

- Gateway Network
- Email Settings

SECURITY

- Auditing
- Users, Roles
- Service Security
- Security Zones

DATABASES

- Connections
- Drivers
- Store and Forward

ALARMING

- General
- Journal
- Notification
- On-Call Rosters
- Schedules

TAGS

Add Connection Step 1: Choose Driver

Select the correct JDBC Driver for the type of database you wish to connect to. If no driver corresponds to your database, go to the Driver Configuration page to add a new driver.

- ☐ Firebird JDBC Driver
The Jaybird JDBC driver for Firebird
- ☐ IBM DB2
The official IBM DB2 JDBC Driver.
- ☐ Microsoft SQLServer JDBC Driver
The Microsoft SQL Server JDBC Driver is a Java Database Connectivity (JDBC) 4.0 compliant driver.
- ☐ MySQL ConnectorJ
The official MySQL JDBC Driver, Connector/J.
- ☒ Oracle JDBC Driver
The Oracle Database JDBC driver.
- ☐ PostgreSQL JDBC Driver
The official PostgreSQL JDBC Driver.

Next >

5. Enter the following information:

Name: **Oracle**

Connect URL: **jdbc:oracle:thin:@localhost:1521:XE** (XE is the System ID in Oracle)

username: **SYSTEM**

password: **oracle** (password is what you entered in during the Oracle server installation, in this example, the password is **oracle**)

Controller | Master

USER MANUAL | SUPPORT

admin | Sign Out

Launch Designer

Ignition
by inductive automation

HOME | STATUS | CONFIGURE

Search...

License Incomplete ? 0:00:00

Reset Trial | View Modules

SYSTEM

Overview

Backup/Restore

Licensing

Modules

Projects

Redundancy

Gateway Settings

NETWORKING

Gateway Network

Email Settings

SECURITY

Auditing

Users, Roles

Service Security

Security Zones

DATABASES

Connections

Drivers

Store and Forward

ALARMING

General

Journal

Notification

On-Call Rosters

Schedules

TAGS

History

Realtime

OPC-UA SERVER

Certificates

Devices

New Database Connection

Main Properties

Name	Oracle
Description	
JDBC Driver	Oracle JDBC Driver
Connect URL	jdbc:oracle:thin:@localhost:1521:test
Username	SYSTEM
Password	*****
Password	*****

6. Click **Create New Database Connection** at the bottom of the form.

The Database Connections page is displayed showing the **Status** of your connection as **Valid**.

Controller | Master

[USER MANUAL](#)
[SUPPORT](#)



[HOME](#)
[STATUS](#)
[CONFIGURE](#)

[admin](#)
[Sign Out](#)

[Launch Designer](#)

SYSTEM

[Overview](#)
[Backup/Restore](#)
[Licensing](#)
[Modules](#)
[Projects](#)
[Redundancy](#)
[Gateway Settings](#)

NETWORKING

[Gateway Network](#)
[Email Settings](#)

SECURITY

[Auditing](#)
[Users, Roles](#)
[Service Security](#)
[Security Zones](#)

License Incomplete ?
0:00:00

[Reset Trial](#)
[View Modules](#)

Database Connections

Successfully updated Database Connection "Oracle"

Name	Description	JDBC Driver	Translator	Status	
MySQL		MySQL ConnectorJ	MYSQL	Valid	delete edit
Oracle		Oracle JDBC Driver	ORACLE	Valid	delete edit

[Create new Database Connection...](#)

Note: For details about a connection's status, see the [Database Connection Status](#) page.

- To display the details about the status of your database connection, see the **Note** on the above window and click on the **Database Connection Status** link.

Connections

[Configuration](#)

Databases

Valid Connections

1 / 1

Total Throughput

0.1^{queries/sec}

Name	Driver	Status	Connections	Throughput	
Oracle	Oracle JDBC Driver	Valid	0 / 8	0.1 queries/sec	Details

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Connecting to PostgreSQL



INDUCTIVE
UNIVERSITY

Connecting to PostgreSQL

[Watch the video](#)

Overview

This page details how to configure a connection to a PostgreSQL database. Note that in some cases you may need to update your JDBC driver, so it is a good idea to consult the table of [supported databases](#).

To Connect Ignition to the PostgreSQL Database

1. From the Gateway, go to the **Configure** section.
2. From the menu on the left, select **Databases > Connections**.
The **Database Connections** page will be displayed.
3. Look for the blue arrow, click on **Create new Database Connection...**

The screenshot shows the Ignition Gateway interface. At the top, there's a dark blue header with the Ignition logo and navigation links like 'HOME', 'STATUS', and 'CONFIGURE'. A green banner across the top indicates 'License Incomplete' with a timer at 0:00:00 and buttons for 'Reset Trial' and 'View Modules'. On the left, a dark blue sidebar contains a search bar and a menu with categories like 'SYSTEM', 'NETWORKING', and 'SECURITY'. Under 'SYSTEM', 'Connections' is selected. The main content area is titled 'Database Connections' and contains a table with columns: Name, Description, JDBC Driver, Translator, and Status. The table is currently empty, showing 'No Database Connections'. Below the table, there is a blue button with a right-pointing arrow and the text 'Create new Database Connection...'. A light blue note box below the button states: 'Note: For details about a connection's status, see the Database Connection Status page.'

4. Select the **PostgreSQL JDBC Driver** and click **Next**.

5. Enter the following information:

- Name: **PostgreSQL**
- Connect URL: **jdbc:postgresql://localhost:5432/test**
- username: **postgres**
- password: password is what you entered in during the Postgres installation.

6. Click **Create New Database Connection** at the bottom of the form.

The Database Connections page will be displayed showing the **Status** of your connection as **Valid**.

7. To display the details about the status of your database connection, see the **Note** on the above window and click on the **Database Connection Status** link.

JDBC Driver

Changed in 7.9.19

In 7.9.19, new installations of Ignition ship with Postgres JDBC driver version 42.3.3. This is notable in cases where a new version of Ignition is restoring from an older gateway backup, as the Extra Connection Properties on any existing Postgres database connection may need to be updated due to the newer driver.

Existing installations upgrading to 7.9.19 do not need to take any action, as the upgrade process does not replace JDBC drivers.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

JDBC Drivers and Translators

Overview

In most cases, the default JDBC drivers and Translator settings in Ignition will not need to be modified. However, there are cases where drivers or translators may need to be modified.

JDBC Drivers and Ignition Upgrades

When upgrading Ignition, JDBC drivers are **not** modified during the upgrade process. This is to prevent database connection issues on upgrade: you should only need to upgrade your JDBC drivers when the database is updated, not when Ignition is upgraded.

New Ignition installations will always use the JDBC drivers that are included with the installer.

Thus running the installer/upgrader on a preexisting Ignition installation will not modify any JDBC drivers, even if newer drivers are included in the installer. This means that if a newer version of an Ignition installer contains a newer JDBC driver, the newer driver will not be used. In this case, you will need to manually update the JDBC drivers.

Adding a JDBC Driver

To add a new JDBC driver to Ignition, do the following steps:

1. In Gateway on the **Configure** section, click on **Databases > Drivers**.
The **Database Drivers & Settings** page is displayed.
2. Find the blue arrow and click on the **Create new JDBC Driver...** link.
3. In the **Name** field, type the full name of the JDBC driver, see the manufacturer's documentation to get the name.
4. In the **JAR File(s)** field, specify the core Java JAR file that contains the driver, as well as any other required JARs.
5. Use the default settings for the following properties:

Driver Defaults and Instructions

Setting	Description
Driver Type	Is the brand of database. This is used for optimizations in the Gateway, if in doubt, select GENERIC.
URL Format	Is a default value for the connect URL. This provides a hint to the format of the connect URL that this driver requires while adding a datasource connection. For example, the hint for the format can be, <code>jdbc:dbtype://host:port/database</code>
URL Instructions	Free form instructions that are shown to help the user to create a connection.
Default Connection Properties	Any additional properties to add by default to the connection string.
Connection Properties Instructions	Tips about which connection properties might be useful.
Default Validation Query	The default query that is used to verify that the connection is available.

SQL Language Comaptibility

Setting	Description
Default Translator	The database translator that is used by default for connections from this driver.

- Click the **Create New JDBC Driver** button, located at the very bottom of the page, to create the new driver.

Upgrading a JDBC Driver

In some cases you may need to upgrade a driver. The steps below detail where this would take place

1. You will need to obtain the new driver. These are typically provided by the same organization that made the database. The driver will be a JAR file.
2. Once you have the new driver, head to your Ignition gateway's **Configure** section, click on **Databases > Drivers**.
3. The **Database Drivers & Settings** page will be displayed. These are the currently configured JDBC drivers on the gateway, and can be modified from the **Edit** button. Click the Edit button for the driver you need to upgrade.
4. You will need to pass in the new driver to the **JAR File(s)** property. Click the **Choose File** button, navigate to the driver, and click **Open**.
5. **[Optional]** Update any other properties. In most cases, you may skip this step. However you may need to update some other properties when a new driver is in place. This step depends on the driver, and what it changes. Refer to the driver's documentation to determine if any connection properties need to be changed. For example, users upgrading to MySQL 8.0 from legacy versions will need to change the **Default Connection Properties** value from
`zeroDateTimeBehavior=convertToNull;` to
`zeroDateTimeBehavior=CONVERT_TO_NULL;useSSL=false;allowPublicKeyRetrieval=true;`
6. Click the **Save Changes** button at the bottom of the page.

Database Translators

Despite the presence of a SQL standard, many database system vary in how they implement or accomplish various tasks. The JDBC driver system tries to hide these differences as much as possible, but unfortunately some differences persist.

The **database translator** system in Ignition navigates these differences as they apply to the system. It provides a way to define certain key operations that are commonly different between database vendors, such as creating auto-incrementing index columns, and the keywords used for different data types.

Translator Management

Database translators are managed in the Gateway from the **Databases > Drivers > Translators** tab. Ignition comes pre-configured with translators for the major supported databases, but you can edit and remove them, as well as create new translators. It is necessary to create a new translator only

when adding a new JDBC driver for a database that does not share syntax with any of the existing translators.

Creating a New Translator

To add a new database translator to Ignition, do the following steps:

1. In Gateway on the **Configure** page, click on **Databases > Drivers**.

The **Database Drivers & Settings** page is displayed.

2. Go to the **Translators** tab, find the blue arrow, and click on the **Create new Database Translator...** link.

The **New Database Translator** page is displayed showing a list of all the translator properties.

3. Define the tokens used with the translator properties on the **New Database Translator** page.
For most of the properties, you need to define special token markers to indicate places where other values are placed. For example, the default **Create Table Syntax** entry looks as follows:

```
CREATE TABLE {tablename} ({creationdef}{primarykeydef})
```

Where `{tablename}`, `{creationdef}`, and `{primarykeydef}` are all tokens that are expanded. `tablename` is replaced directly with the table, `creationdef` is a list of columns, and `primarykeydef` is the phrase created by the **Primary Key Syntax** entry in the translator.

Possible Tokens

Token	Description
<code>tablename</code>	The name of the table being created.
<code>indexname</code>	The name of the index to create, when adding a column index to the table.
<code>primarykeydef</code>	A clause that defines a primary key for a new table.
<code>creationdef</code>	The list of columns to create in the table.
<code>alterdef</code>	A list of columns to add/remove/modify in the table.
<code>columnname</code>	The name of a column.
<code>type</code>	The data type of a column.
<code>limit</code>	The value of the limit clause.

Other Properties

Property	Description
Limit Position	Defines where the limit clause should be placed. Back , the limit is placed at the end of the query. Front , places it directly after the SELECT keyword.
Column Quote Character	All columns are created and accessed with the defined quote, which tells the database to use a specific casing, as well as avoiding collisions between the column name and database keywords.
Supports Returning Auto-generated Keys? / Fetch Key Query	Indicates whether the JDBC driver supports the return of generated keys. If the driver does not support this feature, the Fetch Key Query is used to retrieve the last key.

Data Type Mapping

Type	Description
All data types	The keywords that are used when creating columns of the given types.

4. Click the **Create New Database Translator** button, located at the very bottom of the page, to create the translator.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Store and Forward

The store-and-forward system provides a reliable way for Ignition to store data to the database. In Ignition, systems such as Tag Historian and SQL Bridge (Transaction Groups) use store-and-forward to ensure that data reaches its destination in the database, and is stored in an efficient manner. The store-and-forward system can be configured in a number of ways, offering both memory buffering for performance and local disk caching for safe storage.

NOTE

Store-and-forward engines are automatically created for each Database Connection.

Primary Features and Benefits

The store-and-forward system offers a number of benefits over other systems that log directly to the database, such as:

- **Data loss prevention**
 - Data is removed from the system only when the write to the database has executed successfully.
- **Guaranteed ordering**
 - Data is forwarded in the same order that it arrived, even if a database connection is not currently available.
- **Enhanced performance**
 - By first buffering the data in memory, the store-and-forward system can optimize writes, and prevent the originating systems from blocking. This means that the system is less likely to lose data samples in the event of system slow downs.

Store and Forward Data Flow

Although the system offers settings that can affect the pipeline, by default the data flow occurs as follows:

1. Data is generated in some system.
2. Data is placed in a memory buffer.
3. If not removed from memory buffer in some time (the **Write Time**), or if a certain amount of data accumulates (**Write Size**), it is placed in the local cache.
4. The data sink, based on a database connection, pulls data in first from the local store, and then the memory buffer, based on the **Write Time** and **Write Size** settings under **Forward Settings**.
5. If the data fails to forward, either due to an error in the connection or in the data itself, it is returned to the buffer or cache.
6. If the data errors out too many times, it becomes quarantined.
7. Quarantined data can be managed through the Gateway, and can be deleted or un-quarantined, once the error is resolved.

Understanding the Forward Triggers

Data is forwarded from one stage to the next based on the **Write Time** and **Write Size** triggers. These settings work as an **either/or** manner, meaning that if either of them is surpassed, the data is forwarded. One important point to note is that the Write Size setting influences the transaction size of similar data to be forwarded, and therefore can have a big impact on performance. As a result, the **Write Time** should normally be used as the controlling factor, with the **Write Size** set to something that will provide reasonable transactions, like 100.

Store and Forward for Reliability

The store-and-forward system settings, while seemingly limited, offer a good deal of flexibility in tuning. Different types of situations and goals will likely require different configurations.

When the safety of the data is a concern, the goal is to get the data stored to disk as quickly as possible in order to minimize risk of loss due to a power outage or system failure. The local cache plays a crucial role in this, allowing the system to store data locally for any amount of time until the remote database can accept it. This protects against network failures and database failures, as well.

By setting the **Write Size** and **Write Time** of both the local cache and forwarder to low values, the data spends less time in the memory buffer. While the memory buffer can be set to 0 to bypass it completely, this is not usually recommended, as the buffer is used to create a loose coupling between

the history system and other parts of Ignition that report history. This disconnect improves performance and protects against temporary system slowdowns. In fact, it is recommended that for reliable logging, this value be set to a high value, to allow the maximum possible amount of data to enter the system in the case of a storage slowdown.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Using Store and Forward



INDUCTIVE
UNIVERSITY

Using Store and Forward

[Watch the video](#)

The current status of the Store and Forward system can be viewed from the Status section of the [Gateway Web Interface](#). These pages provide detailed analysis on each Store and Forward engine.

To Monitor the Store and Forward Engine

1. Go to the **Status** section of the Gateway.
2. Click on **Store & Forward** from the left menus.

The **Store & Forward Connections** page will be displayed showing each store-and-forward engine along with the current throughput and capacity of its **Memory Buffer** and **Disk Cache**.

On this page, there are several notable items:

Name	Description
Aggregate Throughput	The aggregated number of records inserted into a database from any engine, per second.
Total Quarantined	The current count of quarantined items across all engines.
Total Dropped	The number of records that have been dropped from all store and forward engines. A record is considered dropped if it can not be added to one of the buffers, such as when a buffer is full, and the engine can no longer accept new records.

Controller | Master

USER MANUALSUPPORT

adminSign Out

Launch Designer

Ignition
by inductive automation

HOMESTATUSCONFIGURE

Search...

SYSTEMS

OverviewPerformanceAlarm PipelinesGateway ScriptsModulesRedundancyReportsSFCsVoice AlarmingTagsEAM TasksTransaction Groups

CONNECTIONS

EAM AgentsDatabasesDesignersDevicesGateway NetworkStore & ForwardOPC Connections

License Incomplete ?0:00:00Reset TrialView Modules

Connections

Store and Forward

Configuration

Aggregate Throughput

23.2per second

Total Quarantined

0

Total Dropped

0

Store and Forward Engines

Name	Memory Buffer	Disk Cache	Quarantined	Activity	
MySQL	7/250	0/25000	0	✓ Available	Details

3. You can click on **Details** under the **Store and Forward Engines** section to refresh and update the displayed values. This page provides in-depth information on the current status of the engine.

Name	Description
Memory Buffer	The number of records entering the Memory Buffer per second. The progress bar shows the percent of the buffer being utilized, along with the current and max number of records.
Disk Buffer	The number of records entering the Disk Buffer per second. Note that a state of "idle" means the engine is able to successfully store all records into the database before the Write Size or Write Time values have been reached. The progress bar shows the percent of the buffer being utilized, along with the current and max number of records.
Database	Shows the number of records pushed from either buffer to the database per second
Quarantined Items	Lists all quarantined items in the engine. Includes the number of occurrences, a description of where the items originated from, and the reason why the record was placed into the quarantine. Provides an opportunity to retry, export, or delete the items.

SYSTEMS

- Overview
- Performance
- Alarm Pipelines
- Gateway Scripts
- Modules
- Redundancy
- Reports
- SFCs
- Voice Alarming
- Tags
- EAM Tasks
- Transaction Groups

CONNECTIONS

- EAM Agents
- Databases
- Designers
- Devices
- Gateway Network
- Store & Forward
- OPC Connections
- OPC-UA Server
- Vision Clients

DIAGNOSTICS

- Execution
- Logs
- Threads

Connections

Configuration

Store and Forward / Connection [MySQL]

Store Details

Memory Buffer

27 /sec

4% (9/250)

Disk Buffer

Idle

0% (0/25000)

Database

0.0 /sec

Quarantined Items

Import quarantine file

Retry All

Delete All

ID	Count	Description	Reason
No items to display			

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Configuring Store and Forward



INDUCTIVE
UNIVERSITY

**Configuring Store and
Forward**

[Watch the video](#)

Store and Forward for High-speed Buffering

When configuring the store-and-forward system for high-speed buffering, you are expecting the case that data will come in quick bursts. By buffering the data, the system can accommodate more information than would be possible going directly against the database.

The key points in configuring a buffering system is to avoid expensive operations like storing and reading from the local cache, and to set the memory buffer large enough to accommodate the expected burst sizes.

Each database connection has its own store and forward settings. Store-and-forward engines are directly correlated to database connections, and are automatically managed so that each connection has an engine defined.

You can create multiple database connections pointing to the same database if you wish to configure multiple store-and-forward engines for different purposes. To configure the Store and Forward engine for your database, do the following steps:

1. Go to the Gateway **Configure** section and select **Databases > Store and Forward**.
The **Store and Forward** page is displayed and you will see a store-and-forward setting for each of your database connections and you can edit these settings.
2. On the **Store and Forward** page, look for **edit** at the far right of the table and click on it to see all the store-and-forward settings.
The settings of a store-and-forward engine define how and when data is moved through the

system. You must understand these settings so that you can carefully set them according to your goals.

Store and Forward Settings

Buffer Settings

Setting	Description
Memory Buffer Size	The number of records that can be stored in the memory buffer, the first stage of the store-and-forward chain. Other settings define when the data will move from the memory buffer forward, this setting only determines the maximum size. If the max size is reached, additional data will error out and be discarded. The memory buffer cannot quarantine data, so if there are errors and the disk cache is not enabled, the data will be lost. If set to 0, the memory buffer will not be used.

Store Settings

These settings apply to the local disk storage cache.

Setting	Description
Disk Cache Enabled	Turns on the hard-disk cache. Data is stored here if it cannot be forwarded in a timely manner. The cache also stores quarantined data (that is, data with errors).
Max Records	The maximum size of the cache. After the max is reached, data is backed up into the memory buffer, and once that is full, it is dropped. A 'record' is an insert or update statement. These statements may be batches, thus it is possible for a single 'record' to impact multiple rows. The default value is 25,000. Note: The disk cache is intended to be a short term storage system. It is not recommended to increase the Max Records beyond 50,000.
Write Size	The number of records that should be accumulated in the memory store before written to the cache. Writing data in blocks can increase performance, but too large of a size increases the risk of data being lost in the event of a power outage or system failure.
Write Time	The max age of records in the memory buffer before they are stored to the cache. This setting is used in combination with the write size in order to give the forwarder the opportunity to retrieve data directly from the memory store and avoid the write to disk entirely.

Forward Settings

These settings govern when data is forwarded to the database. The data is pulled first from the local cache, and then from the memory store. When no data is present in the cahce, it is pulled directly from the memory store.

Setting	Description
Write Size	The number of records that should be accumulated in the memory store before written to the cache. Writing data in blocks can increase performance, but too large of a size increases the risk of data being lost in the event of a power outage or system failure.
Write Time	The max age of records in the memory buffer before they are stored to the cache. This setting is used in combination with the write size in order to give the forwarder the opportunity to retrieve data directly from the memory store and avoid the write to disk entirely.
Enable Schedule	If enable schedule is selected, the forward engine will only be enabled during the times specified by the pattern. The pattern can specify specific times and ranges using a simple syntax.
Schedule Pattern	The schedule is specified as a comma separated list of times or time ranges. You can use the following formats: 24-hour times, that is 8:00-15:00 (for 8am through 3pm) or 21:00-24:00 (9pm through midnight). 12-hour with am/pm (if not specified, 12 is considered noon): 8am-3pm or 9pm-12am Note: When the time period is over, any queued data will remain cached until the next execution period. That is, the forward engine does not run until all data is forwarded.

- Once you made the changes you want, click **Save Changes** at the bottom of the page. This will take you back to the Store and Forward page.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Controlling Quarantine Data



INDUCTIVE
UNIVERSITY

Controlling Quarantine Data

[Watch the video](#)

Quarantined data is data that has erred-out multiple times during attempts to forward it or data that could not be stored because of some configuration issues. It is removed from the forward queue to allow other data to pass. The most common reason for data quarantining is an invalid schema in the database for the data that is being stored. Quarantined data is held indefinitely until the issue is resolved, then you can either delete it or re-insert it into the queue.

To Handle the Quarantined Data

1. From **Status** section of the Gateway, go to **Connections > Store and Forward**.
2. Click the details tab next to a Store and Forward engine that you would like to see the quarantine data for. Here you will see any quarantined data which are listed according to the order they got quarantined. Each set of data has the option to retry it, delete it, or export it for later use. If there are a lot of quarantined records, it may be a good idea to export and delete them so that the store and forward engine won't fill up and drop records.
3. Fix the problem/error you found that caused the quarantine data.
4. Click on **retry**, or if the data had been exported, import the data using the import tool and then **retry**. This way, you can ensure no data gets lost.

Store Details



Quarantined Items

Import quarantine file

Retry All

Delete All

ID	Count	Description	Reason	
1	296	transactiongrouptest{b3_2,t_stamp}	Table 'advtest.transactiongrouptest' doesn't exist	RetryDeleteExport
2	247	sitewidedata{b3_3,t_stamp}	Table 'advtest.sitewidedata' doesn't exist	RetryDeleteExport

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Tags

What is a Tag?

Tags are points of data and may have static values or dynamic values that come from an OPC address, an expression, or a SQL query. The values can be used on screens, in transaction groups, and more.

There are two types of tags; internal and external tags. By default, a fresh Ignition installation will have an internal tag provider - this can be thought of as a standard internal tag database, and stored in the Ignition Gateway. Additionally, it is possible to create external Database-based tag providers, thus turning your SQL database into the tag database, and stored in [Tag Providers](#). This ability opens up some very flexible architectures.

Despite their low initial learning curve, however, Tags offer a great amount of power in system design and configuration. The ability to aggregate tags from a variety of installations in a central SQL database means that you can build widely distributed SCADA systems more easily than ever before, with a high level of performance and relatively easy configuration. Tag User Defined Types (UDTs) provide an object-oriented approach to tag building, allowing you to define parameterized data types, extend and override types, and then rapidly generate instances. A change to the type definition is then inherited by all instances, drastically saving time when making routine changes. The UDT data types are fully supported by Vision templates, which means you can configure templates for your custom data types and take advantage of drag-and-drop binding to rapidly build complex screens.

To learn more about tags, refer to the section on [Understanding Tags](#).

Ignition Tags or PLC Tags?

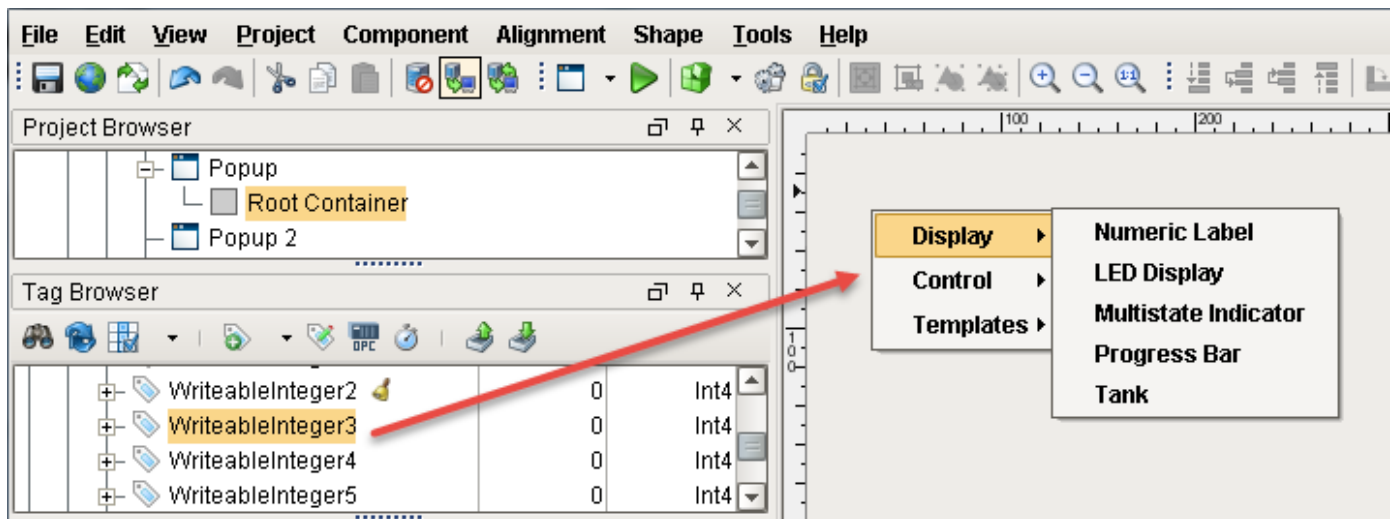
A lot of time you will hear people refer to tags in a PLC as "Tags" as well as tags in Ignition as "Tags." It gets a little confusing sometimes but in this manual if you see "Tags" by itself, we are talking about Tags inside Ignition. Any mentions of tags from a PLC or another OPC Server will be called "PLC Tags" or "OPC Tags."

Main Benefits of Tags

Tags work naturally and easily with Ignition to offer the following features:

- Drag and Drop Screen Design

Using the Vision module, you can drag and drop tags to a window to automatically create new bound components. Drag tags to existing components or properties to quickly bind them to the data. Creating powerful status and control screens is literally just clicks away.



- Object-oriented Design

Use **Tag UDTs** (User Defined Types) to design re-usable, parameterized, and extendable data types. You can create and configure new instance tags in seconds, saving a great amount of time over traditional tag systems.

- Performance and Scalability

Tags offer a great performance on both the Gateway and Client. On the Gateway, the system can support thousands of value changes per second and hundreds of thousands of tags. On Client, Tags improve efficiency with their lightweight subscription architecture. Adding additional Clients creates a nearly negligible effect on the database and the Gateway performance.

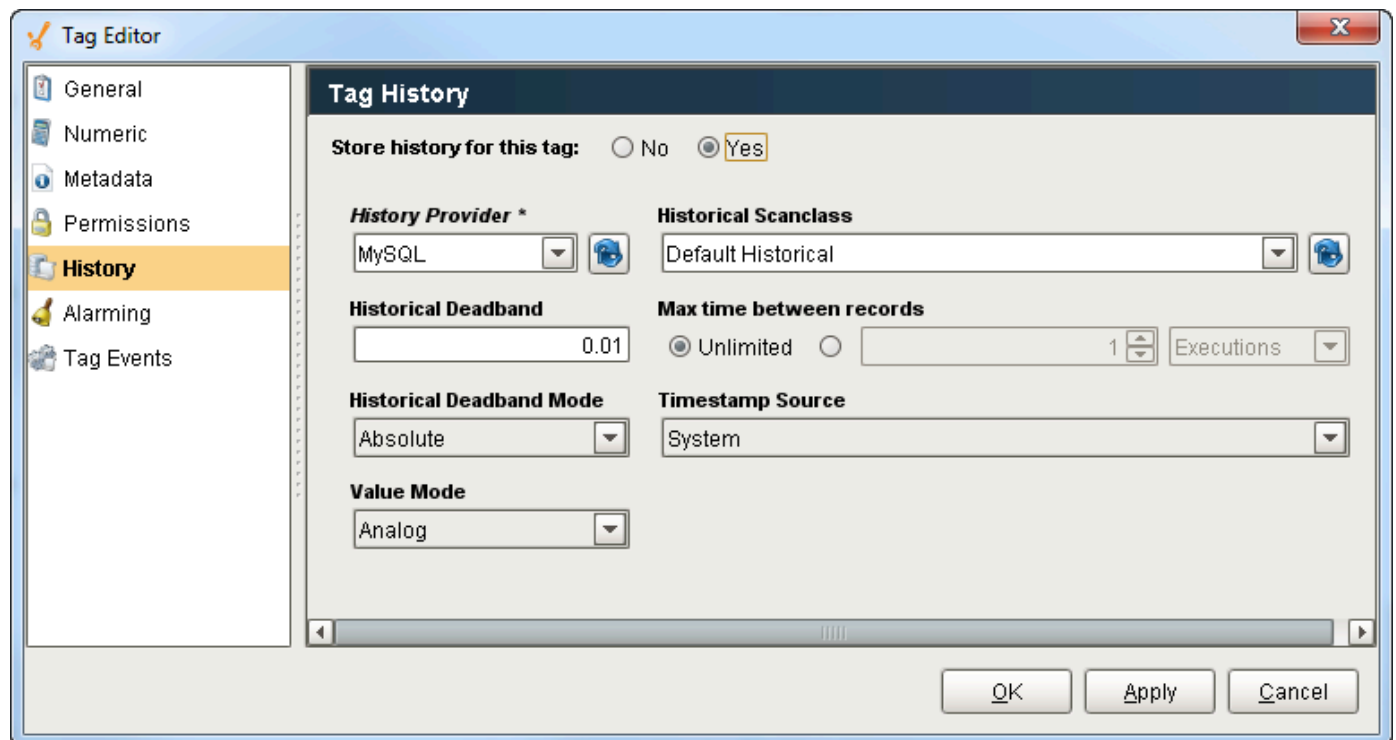
- Integrated Component Feedback

Tags offer a quality and overlay system for Vision module components. If a tag's data quality is anything but good, a component that depends on it gets a visual overlay. Input components display an animated overlay while write pending requests are being written. These features effectively communicate the status of the system at a glance.

Bad Quality OPC_BAD_DATA (0)		Config Error OPC_CONFIG_ERROR (4)	
Not Connected OPC_NOT_CONNECTED (8)		Device Failure OPC_DEVICE_FAILURE (12)	
Sensor Failure OPC_SENSOR_FAILURE (16)		Bad, showing last value OPC_BAD_SHOWING_LAST (20)	
Comm. Failure OPC_COMM_FAIL (24)		Out of Service OPC_OUT_OF_SERVICE (28)	

- Historical Logging

Tags Historian™ makes it easier than ever to store and use **historical data**. When you simply select a check box on a tag, historical data is stored in an efficient format in your SQL database. This data is then available for querying through scripting, historical bindings, and reporting. Also, you can drag-and-drop tags directly onto an Easy Chart to create trends or onto a table to display historical values. Tags Historian's robust querying provides you great flexibility in how you retrieve the data.



- Powerful Alarming Model

Each tag can have any number of alarms configured on it. There are many different **alarm modes** accommodating simple digital alarms, analog high/low value alarms, as well as more specialty alarms like bad data quality and bit-packed alarms. The settings for alarms can bound to other tags, making the alarm configuration dynamic.

Creating Tags

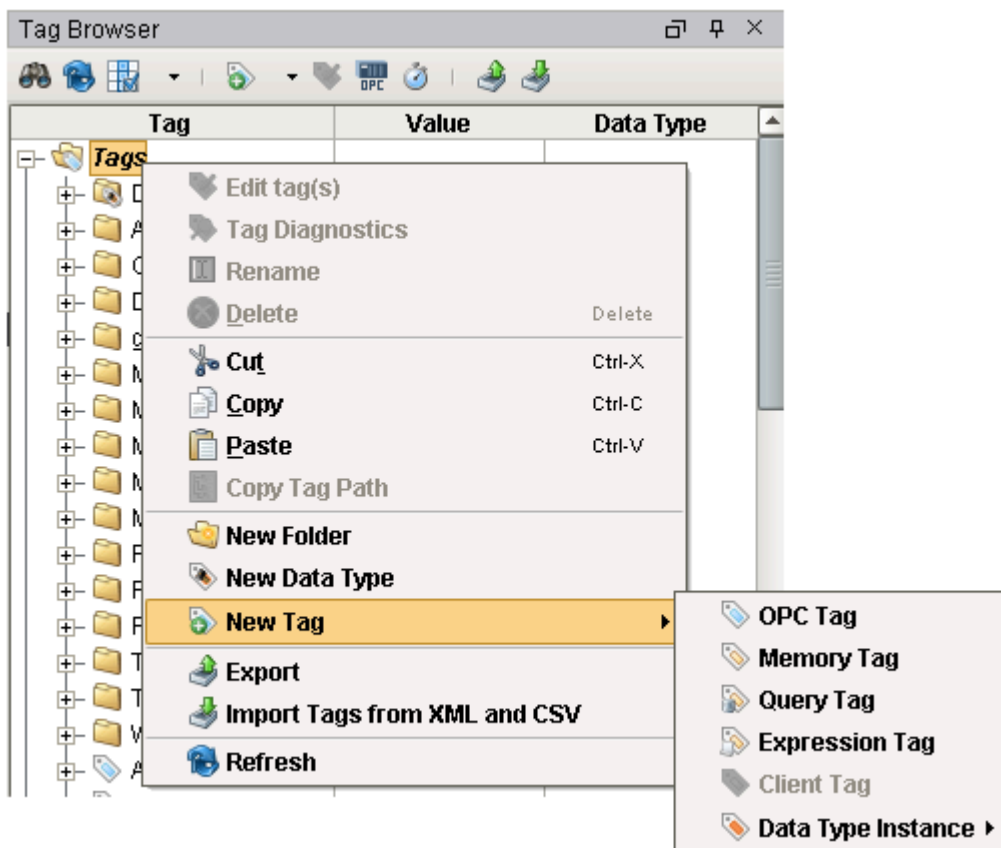
The easiest way to create a tag is from the OPC Browser. But you can also create tags manually using the menu items, or by converting a tag type to a new type.

- Creating From OPC Tags

The easiest and most common way to create Tags is to **drag tags into the Tag Browser** from the OPC Browser. After browsing OPC and finding the tags that you want, simply drag and drop them onto the correct tag provider, and the system will create OPC Tag for each.

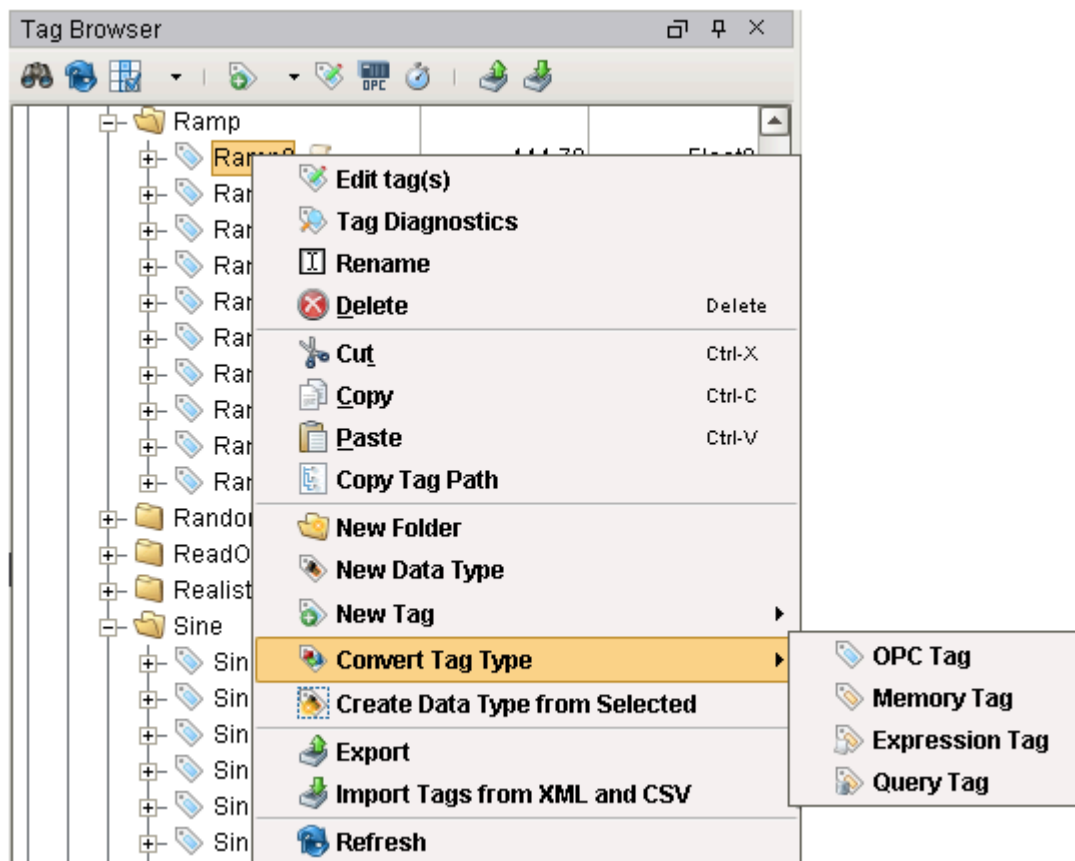
- Creating Tags Manually

The above method only works for OPC tags, and then only for browsable tags. For other types of tags, as well as OPC tags that cannot be obtained through browsing, you can **create Tags manually** in the **Tag Browser**. Click on the **New Tag** icon or right-click on the provider node and select **New Tag**.



- Converting Tag Types

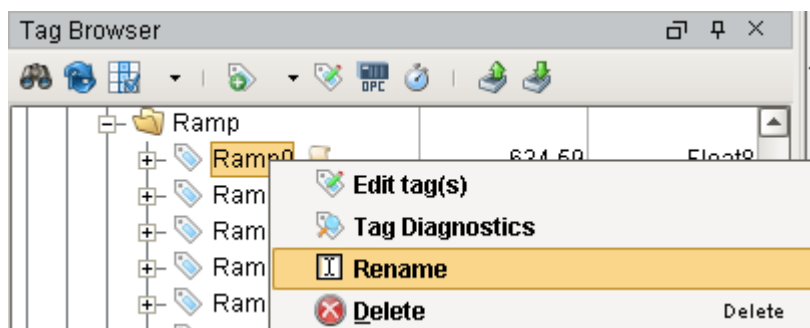
Once created, it is possible to convert a tag to a different type – such as from OPC to Expression. To do this, right-click on the tag, and select **Convert Tag Type**.



Re-naming Tags

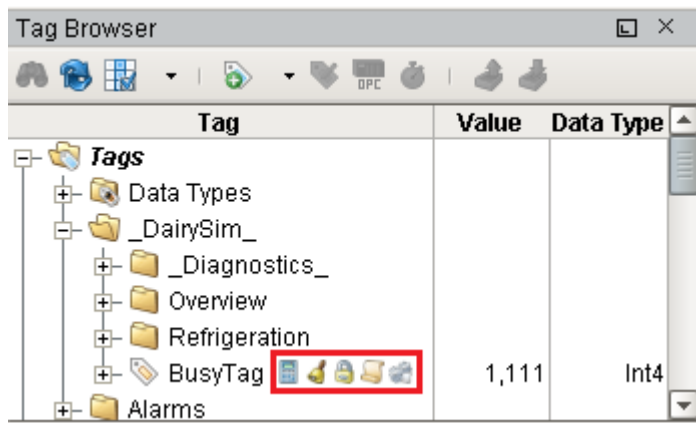
Tags can be named anything ([inside the rules of allowed characters](#)). In other words, it is not necessary that tag's name be related at all to its underlying data source (opc path, for instance). This provides a level of indirection that is convenient for systems whose underlying data storage changes, is based on tag codes (N7, F8, etc.), or for system with many repeat tag structures. By providing tags with meaningful names and arranging them in hierarchical folders, indirect binding can be used to create robust screens that can be used for multiple systems.

See [Keeping Tags Organized](#) for more information about renaming and setting up Tag structures.



Tag Icons

Certain settings or tag configurations are visually represented next to the tag in the Tag Browser.



This icons simply allow you to note some important settings on the tag at a glance. A description of the icons are listed below.

Icon	Setting	Description
	Scaling	The Scale Mode property under the Numeric section of the Tag Editor has been set to a value other than None . The value on the tag will be scaled to some degree.
	Alarming	At least one alarm has been configured on this tag.
	Custom Access Rights	The Access Rights property has been set to Custom . Some users or security zones may be unable to read from or write to this Tag due to the security settings.
	Tag History	This tag has been configured to log data into the Tag Historian system.
	Tag Event Script	At least one Tag Event Script has been enabled on this Tag.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Understanding Tags

Where are Tags Configured?

Tags provide a consistent data model throughout Ignition, and offer the easiest way to get up and running creating realtime status and control systems. Despite their low initial learning curve, however, Tags offer a great amount of power in system design and configuration. The ability to aggregate Tags from a variety of installations means that you can build widely distributed SCADA systems more easily than ever before with a high level of performance and relatively easy configuration.

While the goal of Tags in Ignition is to create an easy yet powerful model, the variety of options and terminology can sometimes make configuration confusing. Tags are created and controlled using both the Gateway and the Designer for configuration.

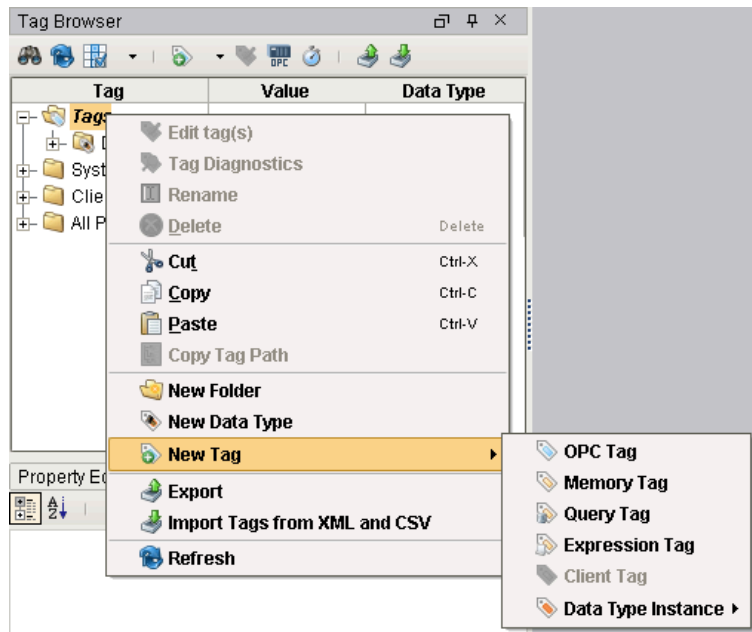
- In the **Designer**, you will create the **Tags**. There are several types of Tags such as an OPC Tags, Memory Tags, and more. Each tag has many properties and other functionality such as alarming, history, etc. Once your tags are created, you can use them in your windows, reports, and more.
- In the **Gateway**, you create and modify **Tag Providers**. You can create these Realtime Providers to store groupings of Tags for use in your projects either internally in Ignition or in an external database. There are also Historian Providers used to store historical data for the tags, but these are automatically created for each datasource you have. These Tag Provider configurations in the Gateway apply globally to all your projects.

Creating Tags

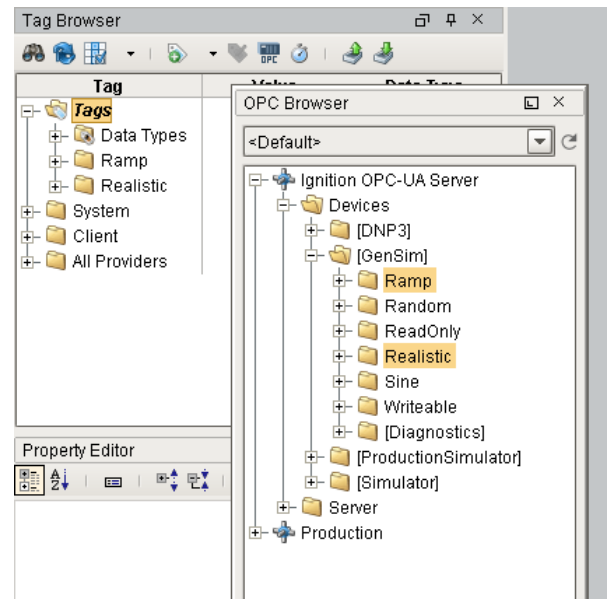
By default, there is one Internal Tag Provider created for you when Ignition is installed. Because of this, you can get started right away with creating Tags.

Tags are created in the Designer, either manual by right-clicking in the Tag Browser and selecting New Tag, or if you already [have a device set up](#), by opening the OPC Browser and dragging tags or folders in. See [Browsing and Creating OPC Tags](#) for more information.

Manually Create Tags



Using the OPC Browser



Working with Tags

Once you have Tags in your Tag Browser, they will automatically start executing. The Designer is live, and you can see your Tags update in real time and write values directly to them. The most commonly used type of tag is an OPC Tag, which gets its value from a device like an OPC Server. You can connect to just about any type of device out there and show the data on screen, as history, and write back whenever you want. Tags make Ignition extremely flexible and easy to show data. See the [Quick Start Guide](#) to get some tags created and in your client.

There are many things you can do with Tags to keep everything organized and give you exactly what you need.

- Learn about the different [types of Tags](#)
- [Organize your Tags](#)
- [Set Metadata properties](#)
- [Set scaling](#)
- [Import/export](#)

Once you have some Tags created, you can edit them to include even more features. You can:

- [Set up Alarms](#)
- [Store Historical data](#)
- [Add Security](#)
- [Add Scripting to Tag Events](#)

In addition to all that, Ignition allows you to **create UDTs** (User Defined Types) out of your Tags to rapidly develop projects with structured objects. This gives you the ability to create a master type then quickly add many instances. UDTs allow you to make a change in one place and see it automatically propagated to every instance.

What are Tag Providers?

Tag Providers and Drivers

At the highest level of configuration is the Tag Provider. A provider is a tag database (a collection of tags) and a name. An Ignition Gateway can have any number of tag providers, and therefore the name is used to distinguish which provider a tag comes from. Tag Providers can be set up with security or even disabled independent of each other.

Every provider holds tags, but not every provider is a Tag driver, or a tag executor. Some tag providers simply make tags available to use, and the tag execution is performed by a different driving provider elsewhere. For example, the Database Provider is a Tag provider that simply watches a tag database stored in an external database. It does not execute tags, it only monitors the values of the tags present. Somewhere else there is a tag driver such as a Driving Datasource Provider or a legacy FactorySQL that is executing the tags and storing the values to the database.

Realtime vs. Historian Providers

As discussed above, all Tags reside in a tag provider and provide realtime values. Additionally, there is the concept of tag historian providers, which can store and query historical data for tags. Each tag can optionally have a historian provider assigned to it to whom it will report value changes for historical storage.

Realtime Providers

The Tags "tag database", or how Tag's tag configuration is stored, can take a few different forms. For internal Tags, the configuration is stored in the Ignition Gateway. With the database form, Tags are stored in a SQL database, outside of the Ignition Gateway. Finally, Remote Tag Providers allow Tag Providers of one Gateway to be accessed on a different Gateway. The different storage mechanisms have different pro and cons, and so it is a good idea to acquaint yourself with each of them.

Internal Tag Provider

The Internal Tags Provider manages the Tags inside the Ignition Gateway. This is the most common method and is optimized for the best scaling and performance.

Remote Tag Provider



INDUCTIVE
UNIVERSITY

Remote Tag Provider

[Watch the video](#)

The Remote Tag Provider works by creating a link from the current gateway to a tag provider on another Gateway in the Gateway Network. This link can be to either an internal provider or a database provider. The catch is that the two Gateways must be linked through the Gateway Network. However, unlike an OPC connection which pulls tags from the OPC server, a remote provider will keep all of the Tag's settings intact. So, things like alarms and history will still be present, and can be used in the Gateway. By default, the remote tag provider will fall under the **Default Security Zone** and be read only.

Database Tag Providers

There are two types of Database Tags providers in Ignition:

- Tag Consuming Provider
- Tag Driving Provider (provided by the SQL Bridge module)

While these two tag providers used to be referred to as external tag providers, they function in the same way that they used to. The Driving Provider will execute Tags as well as make available Tags driven by other Tag Drivers looking at the same database, such as other Ignition Gateways using the Driving Provider, or legacy FactorySQL installations, whereas the Consuming Provider will only read the Tags and will not write values back to the database..

The primary benefit of database providers is that the data is stored in a central database, and is therefore accessible to other data consuming systems that do not have the capabilities to connect to Ignition's OPC-UA server as an OPC-UA client, yet still require the need to consume realtime Tag information.

The negative side to external providers is that all Tag values must be written to the database and then polled for change, which is a less efficient method than internal provider's method. In other words, the inherent architecture of the external model does not scale as well as the internal provider model. For more information about those tables and how they work, see the **External Tag Provider Reference**.

Configuring Realtime Providers

Realtime Tags providers are configured in the Gateway's **Configure** section under **Tags > Realtime**. After installation, the Ignition Gateway will start with an internal provider defined. You can edit its name and settings by selecting **edit** to the right of its entry in the table, or create new providers by selecting **Create new Realtime Tag Provider** below the table.

NOTE

When setting up your Realtime Tag Provider, naming your Provider the same name as the database connection is **not** recommended. Doing so may cause no values to be returned when querying Tag data.

There are four types of **Realtime Tag Providers** that you can choose from:

- **Standard Tag Provider**
- **Remote Tag Provider**
- **Tag Consuming Provider**
- **Tag Driving Provider**

Standard Tag Provider

Tags are stored inside of Ignition and executed by the system.

Property	Description
Name	The name of the provider.
Description	The description of the provider.
Tag Editing Role(s)	Users must belong to one of these roles in order to edit this provider's tags in the Designer. Multiple roles can be specified by separating them with commas. If blank, tag editing for this provider will not be restricted in the Designer.
Default Database	The default database connection to use for expression tags that run SQL queries. All query tags with default database providers selected with run their queries against this database source

Remote Tag Provider

Tag Provider from one Gateway is brought in to another Gateway.

Property	Description
Name	The name of the provider.
Description	The description of the provider
Tag Editing Role(s)	Users must belong to one of these roles in order to edit this provider's tags in the Designer. Multiple roles can be specified by separating them with commas. If blank, tag editing for this provider will not be restricted in the Designer.
Gateway	The name of the Gateway on the Gateway Network that this provider is coming from.
Provider	The name of the provider as it is on the remote Gateway. This does not have to be the same as its name on the new Gateway.
History Access Mode	This setting dictates how tag history is queried for remote tags. Gateway Network will go through the Gateway Network to query history. Database will query the database directly.
History Datasource	The datasource to query when History Access Mode is set to Database.
History Driver	If querying the database directly, this is the gateway name of the remote system. It is used to identify data from that system in the database.
History Provider	If querying the database directly, this is the name of the tag provider on the remote system. It is used along with driver name to identify the correct tags in the database.
Alarms Enabled	If true, alarms configured on the remote gateway will be enabled on the new Gateway.
Alarm Mode	How alarm state should be monitored. In 'queried', state will be queried through the gateway network when necessary. In 'subscribed', the state will be subscribed, and updates will be sent asynchronously. Subscribed provides better performance, but uses more memory.

Tag Consuming Provider

Reads tags stored in an external database and driven by a different instance. Only consumes tags provided by other systems, does not write tag values back to the database.

Property	Description
Name	The name of the provider.
Description	The description of the provider.
Tag Editing Role(s)	Users must belong to one of these roles in order to edit this provider's tags in the Designer. Multiple roles can be specified by separating them with commas. If blank, tag editing for this provider will not be restricted in the Designer.
Database	The database connection where the tag database resides.
Poll rate	The rate in milliseconds at which value and configuration changes will be polled.
Poll overlap	The amount of time, in milliseconds, to overlap the polling time span.

Tag Driving Provider

Stores and executes tags in an external database.

Property	Description
Name	The name of the provider.
Description	The description of the provider.
Tag Editing Role(s)	Users must belong to one of these roles in order to edit this provider's tags in the Designer. Multiple roles can be specified by separating them with commas. If blank, tag editing for this provider will not be restricted in the Designer.
Database	The default database connection where the tag database resides.
Driver name	The unique name of this driver. Since the tags are stored in a central database, there may be multiple providers and drivers operating on them. This name will be used to identify this driver instance, and the tags that it executes. While the driving provider will read all of the tags stored in the database, it will only execute those tags that are assigned to it.
Poll rate	The rate in milliseconds at which value and configuration changes will be polled.
Poll overlap	The amount of time to overlap polls by. If set to 0, the config scan will look for changes only since the last execution. However, on databases that do not support millisecond resolution or are performing less-than-optimally, this could result in missed changes. This setting will expand the window in order to avoid missing these changes.
Enable browsing	Allows remote browsing of the OPC servers available to this driver over TCP/IP. This allows other Gateways to remotely browse and add tags assigned to this driver into the central database.
Browse port	The port to listen on for remote connections. This port must not be in use by any other entity on the machine. Also, each driving provider that wishes to support browsing must have its own port.
Browse address	The IP address/network name that remote Gateways will use when browsing. Therefore, care must be taken that the address is available from the Gateways that will try to connect. Also, since it is used for access by remote systems, it should not be the loopback address (localhost or 127.0.0.1).

Property	Description
	The browse address and port will be stored in the Tags database so that other Gateways can easily look them up. After the settings are configured, you should immediately see the driver name in the OPC browse list for the external provider on other systems looking at the same database. Note: When using remote browsing, make sure that the local firewall has an exception for the port that is used to listen. Otherwise, remote machines will not be allowed to connect.

Enum Reference

Almost every property in Ignition that is a dropdown list has an enumeration that starts at 0 and counts down the list. Below are the definitions of Tag property selections:

Basic Tag fields

Value	Tag Type
0	OPC Tag
1	DB Tag
2	Client Tag
6	Folder Tag

Datatype	String Value	IntegerValue
Byte	Int1	0
Short	Int2	1
Integer	Int4	2
Long	Int8	3
Float	Float4	4
Double	Float8	5
Boolean	Boolean	6
String	String	7
DateTime	DateTime	8
Text	Text	10
Byte Array	Int1Array	17
Short Array	Int2Array	18
Integer Array	Int4Array	11
Long Array	Int8Array	12
Float Array	Float4Array	19
Double Array	Float8Array	13
Boolean Array	BooleanArray	14
String Array	StringArray	15
DateTime Array	DateTimeArray	16
Binary Data	ByteArray	20

Datatype	String Value	IntegerValue
Dataset	DataSet	9
Document	Document	29

Value	Access Rights
0	Read Only
1	Read/Write
2	Custom

Value	Scan Class Modes
0	Direct
1	Driven
2	Leased

Value	Write Response
0	Failure
1	Success
2	Pending

Quality

Value	Data Quality
0	Bad Data from OPC
4	CONFIG_ERROR
8	NOT_CONNECTED
12	DEVICE_FAILURE
16	SENSOR_FAILURE
20	Bad, showing last value
24	COMM_FAIL
28	OUT_OF_SERVICE
32	WAITING
64	UNCERTAIN
68	UNCERTAIN showing last value
80	SENSOR_BAD
84	LIMIT_EXCEEDED
88	SUB_NORMAL
28	SERVER_DOWN
192	Good Data
216	Good, with local override
256	OPC_UNKNOWN
300	Config Error

Value	Data Quality
301	Comm Error
310	Expr Eval Error
330	Tag exec error (fsql)
340	Type conversion error
403	Access Denied
404	Not Found
410	Disabled
500	Stale
600	Unknown (loading)
700	Write Pending

Alarms

Value	Comparison Mode
0	Equal
1	Not Equal
2	Less Than
3	Less Than Equal
4	Greater Than
5	Greater Than Equal

Value	Alert Flags
0x01	Low Exclusive
0x02	Low Infinite
0x04	High Exclusive
0x08	High Infinite
0x10	Any Change
0x20	Low Driven
0x40	High Driven

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Tag Properties



INDUCTIVE
UNIVERSITY

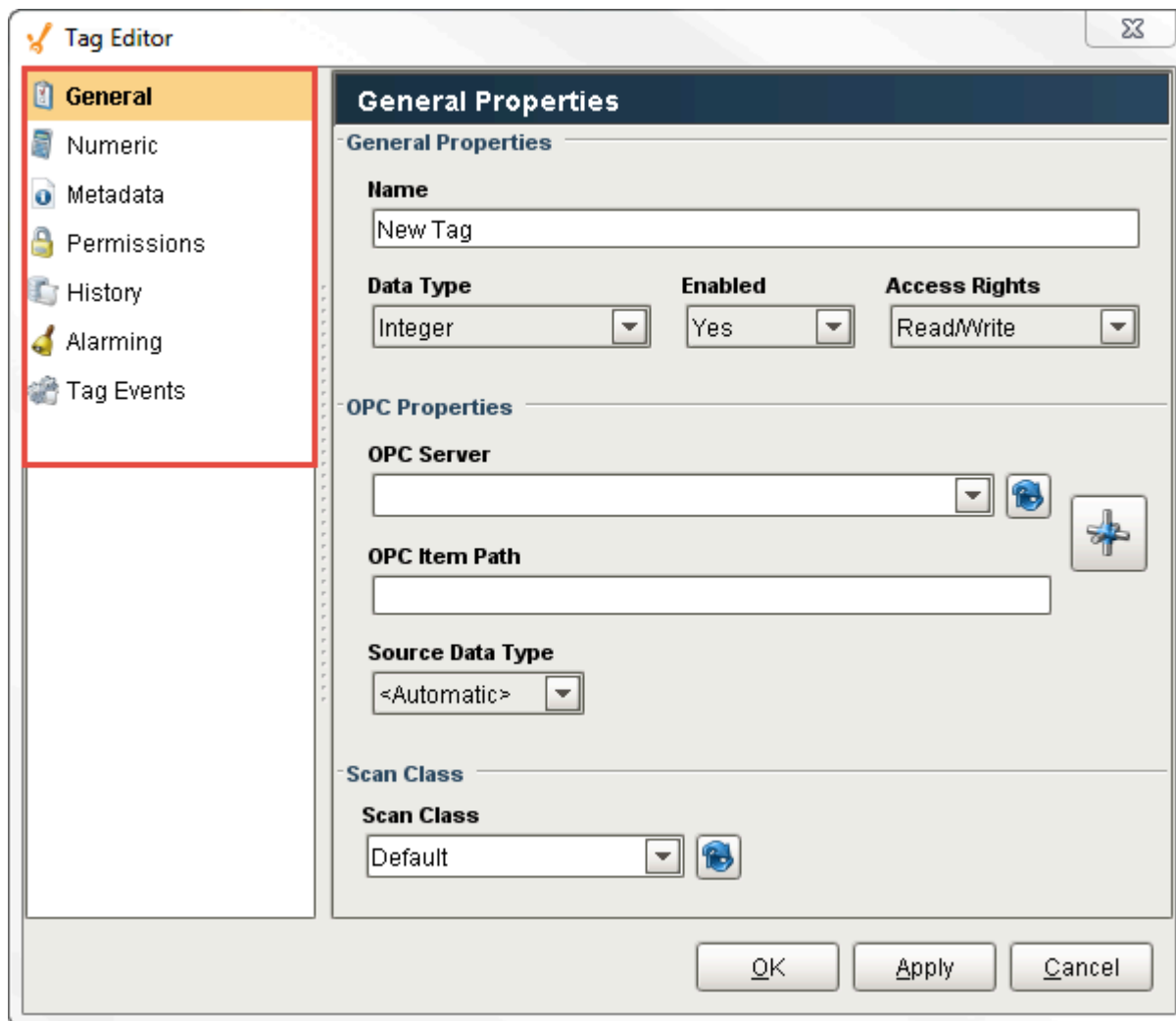
Tags in Ignition

[Watch the video](#)

Tags are points of data and may have static values or dynamic values that come from an OPC address, an Expression, or a SQL query. The values can be used on screens, in Transaction Groups, and more. Additionally, Tags offer a core set of features above and beyond simple values, such as scaling, alarming, and history logging. Depending on the specific **type of Tag**, even more options are available. In general, Tags provide a common interface for tying together many types of data in Ignition.

Tag Configuration in the Designer

To configure a Tag, right-click a Tag and select **Edit Tag**, or right-click the Tags folder in the Tag Browser and use the **New Tag** option to select a new Tag type (The example below is an OPC Tag). The **Tag Editor** window is displayed and you can now set the properties for that Tag. The **Tag Editor** window has the following sections depending on the type of Tag you are creating: General, Numeric, Metadata, Permissions, History, Alarming, SQL Query, Expression, and Tag Events.



General Properties

Properties common to most Tags are as follows:

Property	Binding/Scripting Name	Description	Datatype	Application Tag Type
Name	Name	How the Tag will be presented and referenced in the system. The Tag path will be the provider, the folder structure, and this name.	String	OPC, Query, Expression Derived, Client
Value	Value	The value of the Tag. Can only be modified if the Tag allows value writing and the user has sufficient privileges. This property is conditionally writeable meaning the system.tag.addTag has a parameter specifically for changing the value of a Tag on creation, but system.tag.editTag may not change the Value of Tag. Instead, use system.tag.write .	Object (depends on the datatype of the Tag)	OPC, Query, Expression Derived, Client
Data Type	Datatype	The data type of the Tag. It is important that this be set as correctly as possible with regards to the Tag's	String or Integer See valid values .	OPC, Query, Expression Derived, Client

Property	Binding/Scripting Name	Description	Datatype	Application Tag Type
		underlying data source. The Tags system will attempt to coerce any raw incoming value (for example, from OPC or a SQL query) into the desired type.		
Enabled	Enabled	Whether the Tag will be evaluated by the scan class. If false, the Tag will still be present, but will have a bad quality.	Boolean	OPC, Query, Expression Derived, Client
Access Rights	AccessRights	Specifies the access level allowed to the Tag: Read/Write , Read only , or Custom . If Custom, the Tag will use the permission settings.	N/A The Access Rights of a Tag is not directly editable. However, the value can be modified manually through the Tag Editor window. In regard to scripting, both <code>system.tag.addTag</code> and <code>system.tag.editTag</code> have dedicated parameters that allow Access Rights manipulation.	OPC, Query, Expression Derived, Client

Data Type Valid Values

Datatype	String Value	Integer Value
Byte	Int1	0
Short	Int2	1
Integer	Int4	2
Long	Int8	3
Float	Float4	4
Double	Float8	5
Boolean	Boolean	6
String	String	7
Date	dateTime	8
Byte Array	Int1Array	17
Short Array	Int2Array	18
Integer Array	Int4Array	11
Long Array	Int8Array	12
Float Array	Float4Array	19
Double Array	Float8Array	13
Boolean Array	BooleanArray	14
String Array	StringArray	15
Dataset	DataSet	9

Additional Properties - OPC Tags

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
OPC Server	OPCServer	The server against which to subscribe the data point.	String	OPC
OPC Item Path	OPCItemPath	The path to subscribe to on the server. The point will be subscribed at the rate dictated by the scan class.	String	OPC
Source Data Type	SourceDataType	The data type used by the Tag: Byte, Short, Integer, Long, Float, Double, Boolean, etc..	String or Integer See valid values .	OPC, Query, Expression, Derived

Source Data Type Valid Values

Datatype	String Value	Integer Value
Byte	Int1	0
Short	Int2	1
Integer	Int4	2
Long	Int8	3
Float	Float4	4
Double	Float8	5
Boolean	Boolean	6
String	String	7
Date	dateTime	8
Byte Array	Int1Array	17
Short Array	Int2Array	18
Integer Array	Int4Array	11
Long Array	Int8Array	12
Float Array	Float4Array	19
Double Array	Float8Array	13
Boolean Array	BooleanArray	14
String Array	StringArray	15
Date Array	DateTimeArray	16

Scan Class Properties

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Scan Class	ScanClass	The scan class that will execute the Tag. The scan class dictates the rate and conditions on which the Tag will be evaluated.	String	OPC, Query, Expression, Derived

Additional Properties - Tag in External Providers

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Driver	DriverName	The name of the Ignition Gateway that will be responsible for the execution of the Tag. All other Gateways will monitor the value.	String	OPC
Last Change	LastChange	The last time the Tag was written to or changed. This Tag attribute is not writeable.	Date	OPC, Query, Expression, Derived
Quality	Quality	Identifies if the Tag's data is of good quality, and should be trusted. This Tag attribute is not writeable.	QualityData	OPC, Query, Expression, Derived

Tag Types

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Tag Type	TagType	Tag types that can be created in Ignition are OPC, Memory, Expression, Query and Derived. They are identical in their configurations apart from defining how the value is generated.	Integer See supported values .	OPC, Query, Expression, Derived

Tag Type Supported Values

Tag Type	Integer Value
OPC Tag	0
Memory Tag, Expression Tag, or Query Tag	1
Folder	6
Derived Tag	13

Numeric Properties

The Numeric properties are available to OPC, DB, and Client Tags whose data types are numeric.

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Scale Mode	ScaleMode	If and how the Tag value will be scaled between the source, and what is reported for the Tag.	Integer See valid values .	OPC
Raw Low	RawLow	Start of the "raw" value range.	Double	OPC
Raw High	RawHigh	End of the "raw" value range.	Double	OPC
Scaled Low	ScaledLow	Start of "scaled" value range. Raw low will map to Scaled low for the Tag.	Double	OPC
Scaled High	ScaledHigh	End of "scaled" value range. Raw high will map to Scaled high for the tag.	Double	OPC
Clamp Mode	ClampMode	How values that fall outside of the ranges will be treated. "Clamped" values will be adjusted to the low/high scaled value as appropriate.	Integer See valid values .	OPC
Scale Factor	ScaleFactor	For single parameter modes	Double	OPC

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
		(currently only Exponential Filter), the factor parameter for the equation.	Used when the Scale Mode property is set to Exponential Filter	
Deadband	Deadband	A floating point value used to prevent unnecessary updates for Tags whose values "float" by small amounts.	Double	OPC
Deadband Mode	DeadbandMode	There are two modes to choose from: Absolute or Percent.	Integer See valid values .	OPC

Scale Mode Valid Values

Mode	Integer Value
Off	0
Linear	1
Square Root	2
Exponential Filter	3
Bit Inversion	4

Clamp Mode Valid Values

Clamp Mode	Integer Value
None	0
Low	1
High	2
Both	3

Deadband Mode Valid Values

Deadband Mode	Integer Value
Absolute	0
Percent	1

Metadata Properties



INDUCTIVE
UNIVERSITY

Tag Metadata Properties

[Watch the video](#)

The Metadata properties provide informational properties for a Tag. The values of these fields can be read and modified through scripting, or bound to properties such as Format, Engineering Units, Tooltips, Documentation, and more.

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Format String	FormatString	How the value should be formatted when converted to a string (only applies to numerical data types).	String	OPC, Query, Expression, Derived, Memory
Eng Unit	EngUnit	The engineering units of the value.	String	OPC, Query, Expression, Derived, Memory
Engineering Limits Low	EngLow	The lowest expected value of the Tag.	Double	OPC, Query, Expression, Derived, Memory
Engineering Limits High	EngHigh	The highest expected value of the Tag.	Double	OPC, Query, Expression, Derived, Memory
Engineering Limit Enforcement	EngLimitMode	Dictates how the engineering range should be enforced on the Tag. If not "none", the Tag will change to bad quality ("limit exceeded"), when the value exceeds the specified range.	Integer See valid values .	OPC, Query, Expression, Derived, Memory
Tooltip	Tooltip	The tooltip provides a hint to visual components as to what should be	String	OPC, Query, Expression, Derived, Memory

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
		displayed when the user hovers their mouse cursor over the component that is being driven by the value of this Tag.		
Documentation	Documentation	A freeform text property for information about the Tag.	String	OPC, Query, Expression, Derived, Memory

Engineering Limit Enforcement Valid Values

Limit Enforcement	Integer Value
Off	0
Low	1
High	2
Both	3

Permissions Properties

By default, a Tag's **Access Rights** property is set to **Read/Write**, which means that any user can read the value of the Tag and write to the Tag. **Read-only** mode makes the Tag non-writeable for all users. **Custom** mode allows the Tag to assign Read/Write or Read-Only privileges to individual roles. Any roles not explicitly granted a right by using the Custom Permissions Editor will not be able to read the Tag's value or write to the Tag.

⚠ TAG'S ACCESS WRITES

Make sure your Tag's **Access Rights** (in the General section of the Tag Editor window) is set to **Custom** for these permissions to have an effect. Users must have at least one role in this list in

order to have access to the Tag. If a user has more than one of the roles in the list, the role with the highest privilege will apply.

History Properties

The properties on the **History** tab of the **Tag Editor** window specify if and how the Tag's history will be stored in the Tags Historian system.

Property	Binding/Scripting Name	Description	Datatype	Applica Tag Typ
Store History	HistoryEnabled	Whether the Tag will report its history to the Tags Historian system.	Boolean	OPC, Query, Express Derived Memory
History Provider	PrimaryHistoryProvider	Which Tag Historian data store the Tag will target. A particular Tag can only target one history store. For more information, refer to Historian Providers in Configuring Tag Historian.	String The string name of the Historical Provider	OPC, Query, Express Derived Memory
Historical Scanclass	HistoricalScanclass	The scan class to use to evaluate Tag history. This allows the Tag's history to be stored at a slower rate than the status is updated at.	String	OPC, Query, Express Derived Memory
Historical Deadband	HistoricalDeadband	A deadband that applies only to historical evaluation.	Double	OPC, Query, Express Derived Memory
Max Time between Records	HistoryMaxAgeMode	The maximum amount of time that can pass before a new record is logged for the Tag. The "Max Time between Records" property header	Integer See valid values .	OPC, Query, Express Derived Memory

Property	Binding/Scripting Name	Description	Datatype	Applica Tag Typ
		contains the property, HistoryMaxAgeMode, which can be set using the Radio Button for "Unlimited" number of executions. Note: The HistoryMaxAgeMode and the HistoryMaxAge properties have the same property name, but the two are distinctly separate properties that work together: Unlimited and Number of Executions. To learn more, refer to Max Time Between Records in Configuring Tag Historian.		
Max Time between Records (cont.)	HistoryMaxAge	The maximum amount of time that can pass before a new record is logged for the Tag. The "Max Time between Records" property header contains the property, HistoryAgeMode, which can be set using the Radio Button and the Spinner for a specified number of	Integer	OPC, Query, Express Derived Memory

Property	Binding/Scripting Name	Description	Datatype	Applica Tag Typ
		executions, and the Dropdown for the type of execution (Milliseconds, Seconds, Minutes, etc.). Note: The HistoryMaxAgeMode and the HistoryMaxAge properties have the same property name, but the two are distinctly separate properties that work together: Unlimited and Number of Executions. To learn more, refer to Max Time Between Records in Configuring Tag Historian.		
Historical Deadband Mode	DeadbandMode	There are two modes to choose from: Absolute or Percent.	Integer See valid values .	OPC, Query, Express Derived Memory
Timestamp Source	HistoryTimestampSource	Which timestamp is used for the value of the Tag. Refer to Timestamp Source in Configuring Tag Historian.	Integer See valid values .	OPC, Query, Express Derived Memory

Property	Binding/Scripting Name	Description	Datatype	Applica Tag Typ
Value Mode	InterpolationMode	Value Mode dictates the type of value that the Tag represents, how the deadband will apply to the values, and how interpolation should be performed when querying. • Analog values are values that can change such as temperature or tank level. • Discrete values are values that are either "on" or "off" such as a machine state or "fault" condition. To learn more about Analog and Discrete values, refer to Value Mode in Configuring Tag Historian.	Integer See valid values .	OPC, Query, Express Derived Memory

Max Time between Records Valid Values

Max Time Between Records Mode	Integer Value
Unlimited	0
Executions	1
Milliseconds	2
Seconds	3
Minutes	4
Hours	5
Days	6

Historical Deadband Mode Valid Values

Historical Deadband Mode	Integer Value
Absolute	0
Percent	1

Timestamp Source Valid Values

Timestamp Source	Integer Value
System	0
Value	1

Value Mode Valid Values

Value Mode	Integer Value
Discrete	0
Analog	3

UDT Properties

User Defined Types (UDTs) are essentially data templates that can be configured to easily control multiple instances. UDTs can hold multiple other basic types of Tags (Integers, Strings, etc.) as well as other UDTs. Changes made to the parent UDT will automatically be propagated to all children.

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
UDT Parent Type	UDTParentType	UDTs are similar to data templates where a particular structure of a Tag is defined. A UDT can be created as if it were a single Tag to create Tag instances.	String States the UDT Definition associated with a UDT instance	UDT Instance

Alarming Properties

Tags have the ability to define any number of alarms. Each alarm is a condition that will be evaluated when the value of the Tag changes. When the condition becomes true, the alarm is said to be active. When it becomes false, the alarm is said to be cleared.

Here is a list of Alarm Properties that you can edit or bind when creating or editing a Tag.

Main

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Name	name	Each alarm has it's own name. For example, if the Tag is representing a level, the alarm name might be High Level .	String	OPC, Query, Expression, Derived, Memory
Enabled	enabled	This boolean determines whether or not the alarm will be evaluated. A disabled alarm's condition will not be evaluated, and thus will not generate any alarm events.	Boolean	OPC, Query, Expression, Derived, Memory
Priority	priority	An alarm's priority can affect how is appears in an Alarm Status table, or can affect how it is escalated through a pipeline. The priorities, which can be referenced by their integer equivalent in scripts and expressions, are: Diagnostic [0] , Low [1] , Medium [2] , High [3] , Critical [4] .	String	OPC, Query, Expression, Derived, Memory
Timestamp Source	timestampSource	Chooses where the timestamp for the alarm event should come from: the system time of when the event was generated, or the	Integer See valid values .	OPC, Query, Expression, Derived, Memory

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
		timestamp of the value that tripped the event.		
Label	label	New in 7.9.8 An optional name that will be used for display purposes. Provides a dynamic alternative to the static name property. If left blank, the name will be used.	String	OPC, Query, Expression, Derived, Memory
Display Path	displayPath	This is a string value that will be used to display the alarm to operators. If this is blank, the operator will see the path to the Tag instead. Please use the forward-slash character to separate hierarchy levels in this path, for example: East Area/Boilers/Boiler5	String	OPC, Query, Expression, Derived, Memory
Ack Mode	ackMode	Dictates how acknowledgement works for the alarm. Unused - Acknowledgement will not be used for this Tag, and any alarm that is generated will automatically be marked as acknowledged.	String	OPC, Query, Expression, Derived, Memory

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
		• Auto - The alarm is acknowledged automatically when the alarm becomes cleared. • Manual - The alarm is never set to be acknowledged by the system, and it is up to the user to manually acknowledge alarms.		
Notes	notes	A place for any free-form documentation about the alarm that can be displayed to operators.	String	OPC, Query, Expression, Derived, Memory
Ack Notes Required	ackNotesReqd	If this setting is true, the operators will be unable to acknowledge this alarm without entering some notes.	Boolean	OPC, Query, Expression, Derived, Memory
Shelving Allowed	shelvingAllowed	If this setting is true, the shelving feature will be unavailable for this alarm.	Boolean	OPC, Query, Expression, Derived, Memory

Timestamp Source Valid Values

Timestamp Source	Integer Value
System	0
Value	1

Alarm Mode Settings

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Mode	mode	This setting controls what condition this alarm is evaluating. The available modes are as follows: • Equal - Active when the Tag's value equals the alarm's setpoint. • Not Equal - Active when the Tag's value does not equal the alarm's setpoint. • Above Setpoint - Active when the Tag's value is above the alarm's setpoint. • Below Setpoint - Active when the Tag's value is below the alarm's setpoint. • Between Setpoints - Active when the Tag's value is between the low and high setpoints. If any change is true, an event will be generated for each value change between the setpoints. • Outside Setpoints - Active when the Tag's value falls outside the low and high setpoints. If any change is true,	String	OPC, Query, Expression, Derived, Memory

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
		an event will be generated for each value change outside the setpoints. • Out of Range - The same as Outside Setpoints, but uses the Tag's Engineering High and Engineering Low as the high and low setpoints. • Bad Quality - Active if the Tag value becomes a bad quality, for example, on communication loss. • Any Change - An alarm event is generated every time the Tag value changes. Note that this alarm will never be "active" because each active event is paired with a matching clear event, instantly. • Bit State - This alarm mode is used to alarm when a specific bit out of an integer Tag becomes high. You must specify which bit position to use, with zero being the least significant bit. The On Zero property is used to invert the logic and		

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
		alarm when the bit is low. • On Condition - This free-form alarm mode is used for when you want to specify the condition using an expression or another Tag. To do this, bind the "Is Active" property to an appropriate expression or Tag.		
Setpoint / Low Setpoint	setpointA	The value used to initiate an alarm. If the alarm mode calls for two setpoints, this is the low value.	Double	OPC, Query, Expression, Derived, Memory
Inclusive / Low Inclusive	inclusiveA	If true, the Setpoint or Low Setpoint value is used inclusively for the condition to alarm.	Boolean	OPC, Query, Expression, Derived, Memory
High Setpoint	setpointB	The high value used to initiate an alarm when the alarm mode calls for two setpoints. Available for modes: Between Setpoint, Outside Setpoints.	Double	OPC, Query, Expression, Derived, Memory

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
High Inclusive	inclusiveB	If true, the High Setpoint value is used inclusively for the condition to alarm. Available for modes: Between Setpoint, Outside Setpoints.	Boolean	OPC, Query, Expression, Derived, Memory
Any Change	anyChange	If true, will alarm on each value change given the alarm mode conditions are met. Note: This alarm will NEVER be "active" because each active event is paired with a matching clear event, instantly. Available for modes: Above Setpoint, Below Setpoint, Between Setpoints, and Outside Setpoints.	Boolean	OPC, Query, Expression, Derived, Memory
On Zero	bitOnZero	If true, will alarm when the specified bit is not high (when the bit is 0). Available for modes: Bit State.	Boolean	OPC, Query, Expression, Derived, Memory
Bit Position	bitPosition	The position of the bit, starting at 0 that will be watched. Available for modes: Bit State.	Integer	OPC, Query, Expression, Derived, Memory

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Is Active	activeCondition	When this property is active, the alarm will be active. Typically has a binding of some sort that will be used to determine when the alarm goes active. If the expression evaluates to True, the alarm is active. If the expression evaluates to False, the alarm is not active. Available for mode: On Condition.	Boolean	OPC, Query, Expression, Derived, Memory

Deadbands and Time Delays

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Deadband	deadband	The value for the deadband, interpreted according to the Deadband mode. Note that all alarms are only evaluated after the Tag's value changes, which means that the Tag's own deadband will be considered first. When the deadband is positive, an active alarm condition needs to clear its setpoint(s) by the amount of the deadband for the alarm to clear. For example, suppose you had a Between Setpoints alarm with a low setpoint of 50 and a high setpoint of 70, and with a deadband of 2. The alarm will go active if the value is between 50 and 70, but will only clear if the value falls below 48 or rises above 72.	Double	OPC, Query, Expression, Derived, Memory
Deadband Mode	deadbandMode	Defines how the deadband value is	Integer	OPC, Query,

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
		used. • Absolute - The deadband setting is considered to be an absolute value. • Percent - The actual deadband is calculated as a percent of the Tag's engineering unit span.	See valid values .	Expression, Derived, Memory
Active Delay	timeOnDelaySeconds	The time, in seconds, before the alarm will be considered active after the alarm's condition becomes true. Also known as a "rising edge time deadband."	Double	OPC, Query, Expression, Derived, Memory
Clear Delay	timeOffDelaySeconds	The time, in seconds, before an active alarm will be considered clear after the alarm's condition becomes false. Also known as a "falling edge time deadband."	Double	OPC, Query, Expression, Derived, Memory

Deadband Mode Valid Values

Alarming Deadband Mode	Integer Value
Absolute	0
Percent	1

Notification

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Active Pipeline	activePipeline	The name of an alarm notification pipeline to put this alarm into when it becomes active in order to send out active alarm messages. Many alarms may share a single pipeline.	String	OPC, Query, Expression, Derived, Memory
Clear Pipeline	clearPipeline	The name of an alarm notification pipeline to put this alarm into when it becomes cleared in order to send out cleared messages.	String	OPC, Query, Expression, Derived, Memory
Ack Pipeline	ackPipeline	The name of the alarm notification pipeline to put this alarm into when the alarm has been acknowledged.	String	OPC, Query, Expression, Derived, Memory

Email Notification Properties

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Custom Subject	CustomEmailSubject	A string that will be used as the subject line of an email notification message. If blank, the message settings defined on the notification block that sent the email out will be used instead.	String	OPC, Query, Expression, Derived, Memory
Custom Message	CustomEmailMessage	A string that will be used as the body of this alarm's email notification message. If blank, the message settings defined on the notification block that sent the email out will be used instead.	String	OPC, Query, Expression, Derived, Memory

SMS Notification Properties

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
Custom Message	CustomMessage	If specified, will be used for the SMS message. If blank, the message defined in the notification block will be used.	String	OPC, Query, Expression, Derived, Memory

Associated Data

Property Name	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
User Defined Data		Associated Data are custom alarm properties that can be added to any alarm. These properties will often be bound to other Tags that represent associated contextual data that may be related to the alarm. A snapshot of the values of these properties will be taken when the alarm becomes active. These values will be attached to the alarm event as it moves through the rest of the alarming system, meaning that the values will be available from the alarm status system, the alarm journal system, and in the alarm notification system.	String	OPC, Query, Expression, Derived, Memory

Binding

Many alarm properties are bindable, which means they can be bound to other Tags in the system, or expressions. For example, you might bind the enabled property to another Tag which represents whether or not your process is running, thereby, disabling the alarm when production is stopped. Or, you might bind the setpoint of an alarm to a Tag that operators can manipulate, thereby, letting the setpoint be changed at runtime.

To bind a Tag, simply click on the binding icon () and the binding UI will slide in from the right. From here, you can select the binding type (No Binding, Tag, Expression, or UDT Parameter, if applicable). Note, that the expression can reference many useful values such as the Tag's value and other settings of the alarm. When you configured the binding to your liking, click on the **Back** button.

Expression/SQL Properties

Query Tags and Expression Tags have the ability to use an SQL query or expression (respectively) as their value instead of an OPC item path. This can be used to select information from the database or create your own formulas to manipulate other Tag values.

Expression Properties

In Expression Tags, the Tag can use all of the features available in the expression language. It can refer to other Tags, and use operators and functions to calculate a value for the Tag.

SQL Query Properties

In SQL Query Tags, the Tag's value will be the result of the specified SQL query. The query can be any valid query, but should result in only one value. Note, that insert and update queries can be used, and will often result in an integer value, so the Tag's data type should be set accordingly.

Like SQL Query bindings in the Vision module, the queries for Tags can refer to other Tag values. The values of referenced Tags will be inserted as literal text in the query before being sent to the database.

Client Tags

Client Tags have the ability to be used as either Expression or SQL Query Tags. There is an **Expression/SQL** page in the Tag editor that allows you to select what type it is.

Query/Expression Attributes

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
OPC Server	OPCWriteBackServer	The server against which to subscribe the data point.	String	Query, Expression
OPC Item Path	OPCWriteBackItemPath	The path to subscribe to on the server.	String	Query, Expression
Query	Expression	Text area to build your query or expression.	String This is the code used by the Tag: either a SQL Query for Query Tags, or an Expression for Expression Tags.	Query, Expression
Query Type	QueryType	When the TagType property is set to 1, this property determines if the Tag should be a Memory, Expression, or Query Tag.	Integer See supported values .	Query, Expression, Memory
Datasource	SQLBindingDatasource	The default data source of	String	Query

Property	Binding/Scripting Name	Description	Datatype	Applicable Tag Type
		the Tag provider.		

Query Type Supported Values

Tag Type	Integer Value
Memory Tag	0
Expression Tag	1
Query Tag	2

Tag Events

Each Tag has the option to have **Tag Event Scripts** on it. When you edit a Tag, you can navigate to the Tag Events section to see a list of all of the Tag scripts. You can then select which event you would like to write a script for. You can even write a script for multiple events if you like.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Types of Tags



INDUCTIVE
UNIVERSITY

Types of Tags

[Watch the video](#)

There are several different types or scopes of Tags you can create in Ignition: Gateway Executed Tags, System Tags, and Client Tags. Each Tag belongs to a scope, and each scope plays a role in how Tags behave. All these Tag types are available in the Tag Browser.

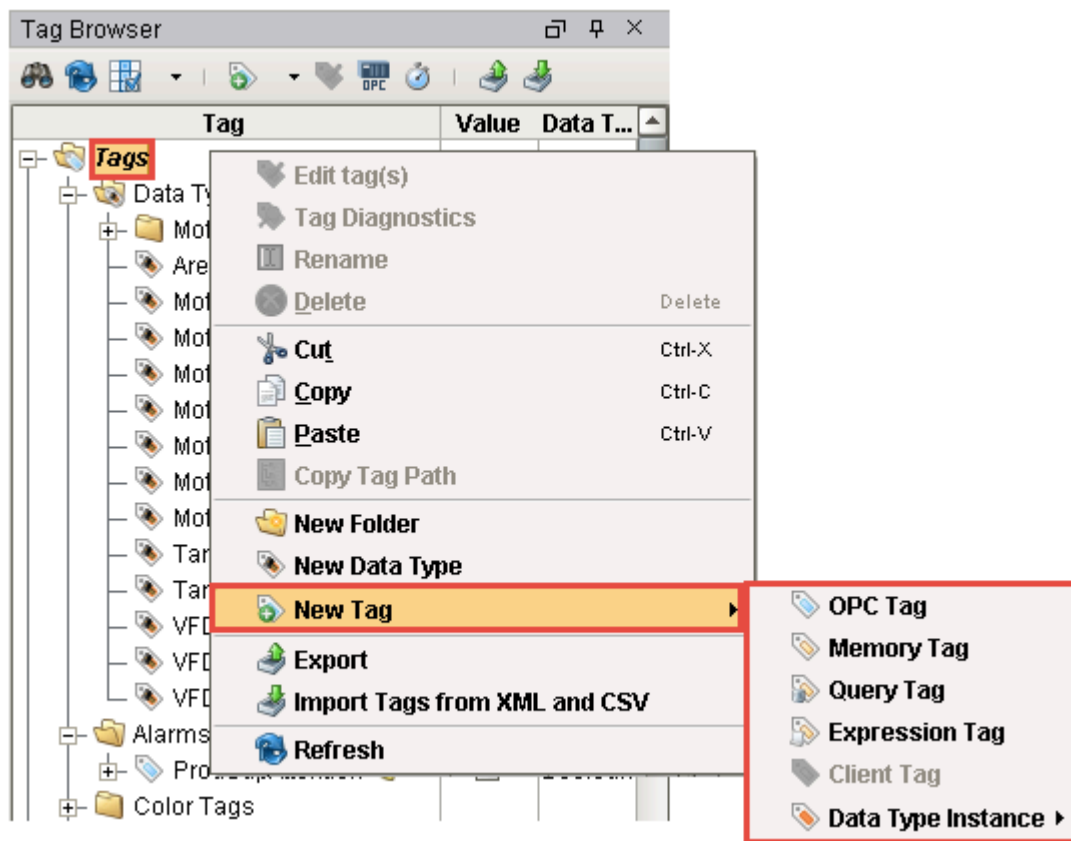
When discussing "Tags", we commonly mean Gateway Executed Tags, but System and Client Tags also play an important role in the overall design of a project.

Let's examine the different types of Tags.

Gateway Executed Tags

Tags executed in the Gateway support all of the primary features of Tags: scaling, alarming, history, and role-based permissions. These Tags are all Gateway scoped, and the values of the Tags are shared among all running clients. They are identical in their configurations, apart from defining how the value is generated.

You can create each of these Tags in the **Tag Browser** by right clicking on the Tags folder, scroll down to **New Tag** and select the desired Gateway Executed Tag.



OPC Tags

An OPC Tag is an OPC server and address which drives values. The OPC server is typically a PLC or device, such as Ignition's OPC-UA server, or any third party OPC-UA server.

In the **Tag Browser**, double click on any existing OPC Tag, to see the the **OPC Server** name and **Item Path**.

Tag Editor

General

- Numeric
- Metadata
- Permissions
- History
- Alarming
- Tag Events

General Properties

General Properties

Name
WriteableInteger1

Data Type
Integer

Enabled
Yes

Access Rights
Read/Write

OPC Properties

OPC Server
Ignition OPC-UA Server

OPC Item Path
[Generic]_Meta:Writeable/WriteableInteger1

Source Data Type
<Automatic>

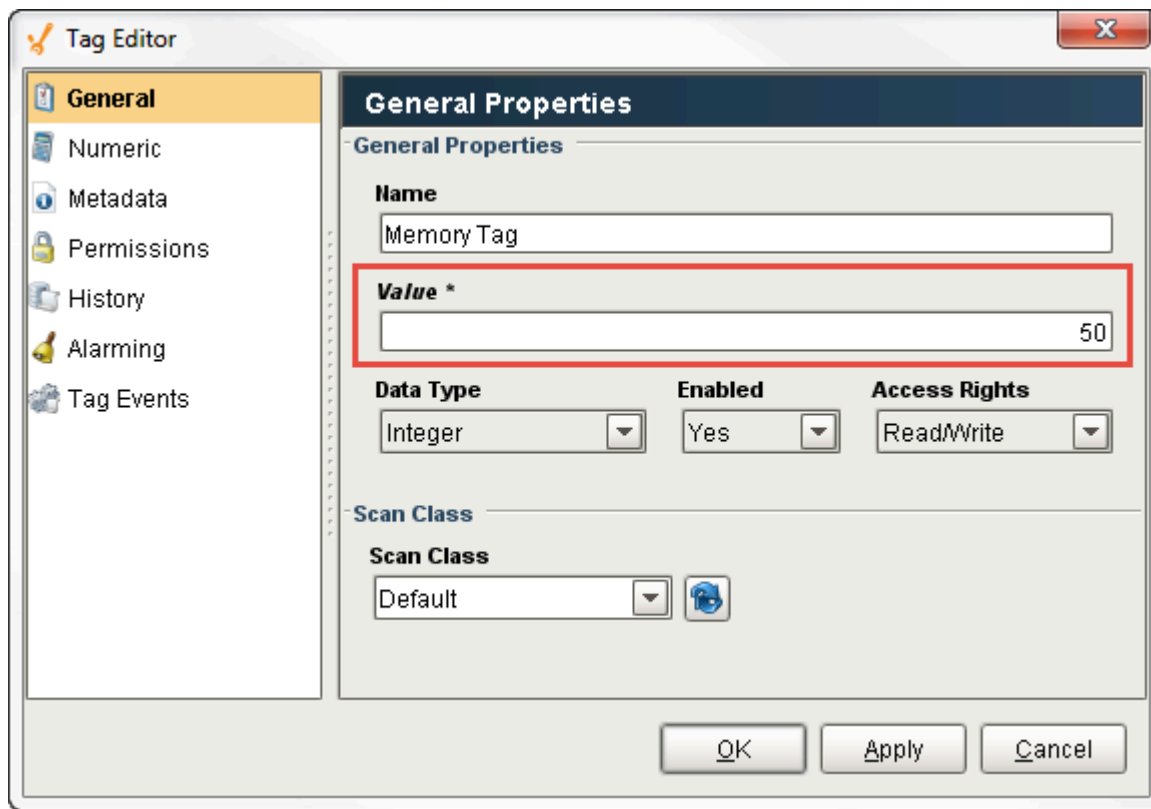
Scan Class

Scan Class
Default

OK Apply Cancel

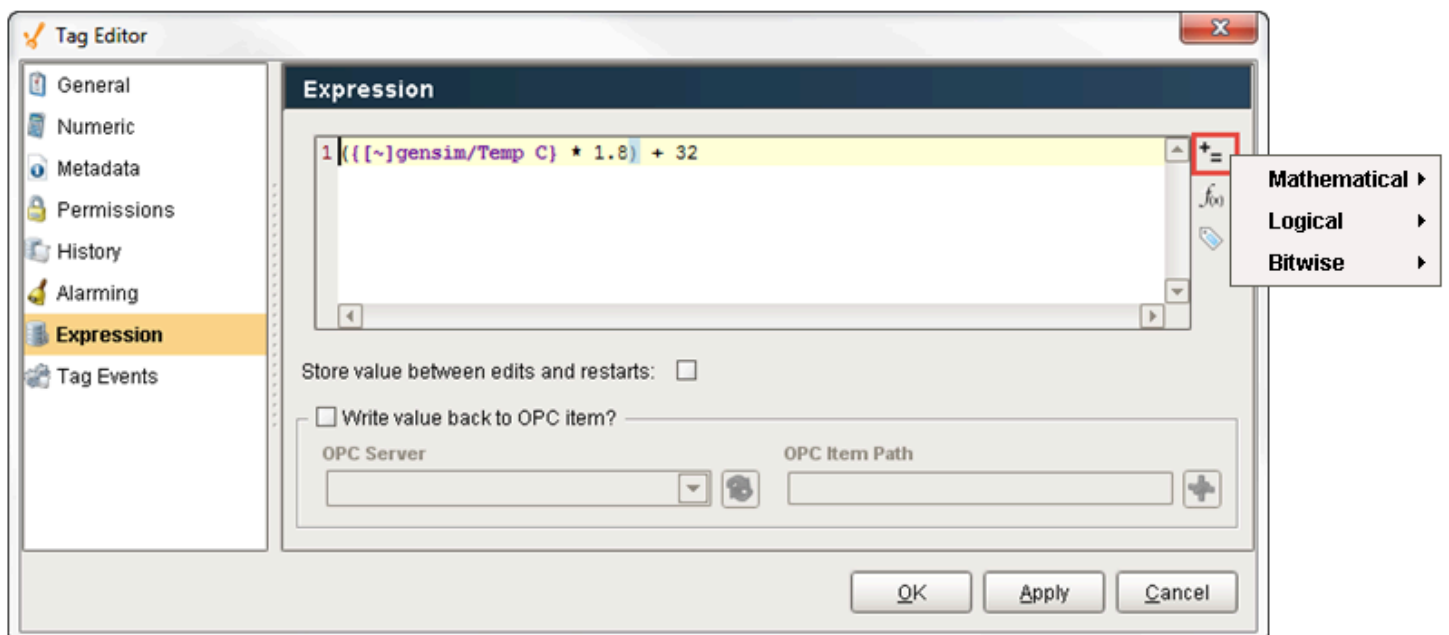
Memory Tags

The Memory Tags are simply values. The value is specified during configuration, and is stored when written (if the Tag allows writing).



Expression Tags

The Expression Tags are driven by an expression, and its values are derived from a calculation. The expression syntax is the same as for property bindings, and allows mathematical operations, references to other Tags, logic operations, functions, and more. This expression example converts Celsius to Fahrenheit.



SQL Query Tags

The SQL Query Tags execute a SQL Query, whose result provides the value for the Tag from the SQL database. Like an SQL binding in Vision, SQL Query Tags can reference other Tags to build dynamic queries. This SQL query example is getting the average of Ramp1 from the History table.

! INFO

Each query Tag will run a separate query, so large numbers of this type of Tag will **dramatically** increase the number of queries running against the database. It is highly recommended to attempt to optimize and restrict the number of query Tags: instead of having several query Tags pulling values from the same table, have one query Tag configured to a Dataset data type that returns large portions of the table, and use several expression Tags to extract values from the individual elements in the dataset.

The screenshot shows the 'Tag Editor' window with the 'SQL Query' tab selected in the left sidebar. The main panel is titled 'SQL Query' and contains a text area with the SQL query: `SELECT AVG(ramp1) FROM ramp_history`. Below the text area, there are two dropdown menus: 'Datasource' set to 'MySQL' and 'Query Type' set to 'Auto Detect'. There are also checkboxes for 'Store value between edits and restarts' (unchecked) and 'Write value back to OPC item?' (unchecked). At the bottom, there are fields for 'OPC Server' and 'OPC Item Path', both currently empty, with a '+' button next to the 'OPC Item Path' field. At the very bottom of the window are 'OK', 'Apply', and 'Cancel' buttons.

Complex Tags (UDTs)

Complex Tags are created out of standard Tag types, but offer a variety of additional features. In simple terms, you can think of them as a way to create "data templates", where a particular structure of Tags is defined, and can then be created as if it were a single Tag. This UDT example shows two Motor instances, the data type Motor, and all the Tags that make up the structure (i.e., AMPS, HI SP, HOA, RUN_CMD, and STATUS).

Tag Browser			
Motor 1 AMPS 30.67 Float4 HI SP 60 Int4 HOA 1 Int2 RUN_CMD 11 Int2 STATUS 1 Int2			
Motor 2 AMPS 7.41 Float4 HOA 1 Int2 RUN_CMD 2 Int2 STATUS 2 Int2			

Derived Tags

New in 7.9.2 A Derived Tag is an abstracted Tag that refers to another Tag. They are similar conceptually to Expressions Tags that reference another Tag, but Derived Tags have some additional functionality. Namely, they can write back to the referenced Tag.

Source Tag Path

The Source Tag Path property of a Derived Tag depicts which Tag the Derived Tag should reference. This is similar to the OPC Item Path on an OPC Tag, except it points to an Ignition Tag.

Aside from editing the Tag from the Designer, the Source Tag Path on a Derived Tag may be modified by writing directly to the **SourceTagPath** property on the Tag, or using a Tag Binding on a Vision component.

Tag Browser

Derived Example

Derived Tag

Value

AlarmActiveAckCol

45

45

0

Int4

Integer

FormatString

LastChange

Name

Quality

SourceTagPath

Tooltip

###0.##

2017-12-31 2:15:07 PM

Derived Tag

Good

[default]Folder\Temp F

String

Date

String

DataQuality

String

Additionally, the **SourceTagPath** property may be changed through scripting:

```
#Example of writing to the SourceTagPath attribute via Python Script
system.tag.write("Derived Example/Derived Tag.SourceTagPath", "[.]Folder/New
Source Tag")
```

! INFO

The Derived Tag will update at the rate of the source Tag.

Value Derivation

Instead of a Numeric section, Derived Tags have a Value Derivation section accessible in the Tag Editor. This section allows you to leverage Expressions when reading from or writing to the source Tag, which provides a greater degree of customization than the traditional Numeric scaling settings available to OPC Tags.

The screenshot shows the 'Tag Editor' window with the 'Value Derivation' tab selected in the left-hand navigation pane. The main area is divided into two sections: 'Read Expression' and 'Write Expression'. The 'Read Expression' section has a text box containing '1 {source}' and a description: 'Use the reference {source} to access the originating value.' The 'Write Expression' section has a text box containing '1 {value}' and a description: 'Use {value} for the incoming write value, and {source} for the underlying source value.' Both text boxes have a yellow highlight and a small 'f(x)' icon to the right. At the bottom of the window are 'OK', 'Apply', and 'Cancel' buttons.

Tag Editor

General

Value Derivation

Metadata

Permissions

History

Alarming

Tag Events

Value Derivation

Read Expression

Use the reference {source} to access the originating value.

1 {source}

Write Expression

Use {value} for the incoming write value, and {source} for the underlying source value.

1 {value}

OK Apply Cancel

Read Expression	Determines what value should appear on the Derived Tag. The Current Value of the source Tag may be referenced with the <code>source</code> reference.
Write Expression	When writing to the Derived Tag, this expression determines what value should be written to the source Tag. The current value of the source Tag may be referenced with the <code>source</code> reference. The raw value passed to the Derived Tag may be referenced with the <code>value</code> reference.

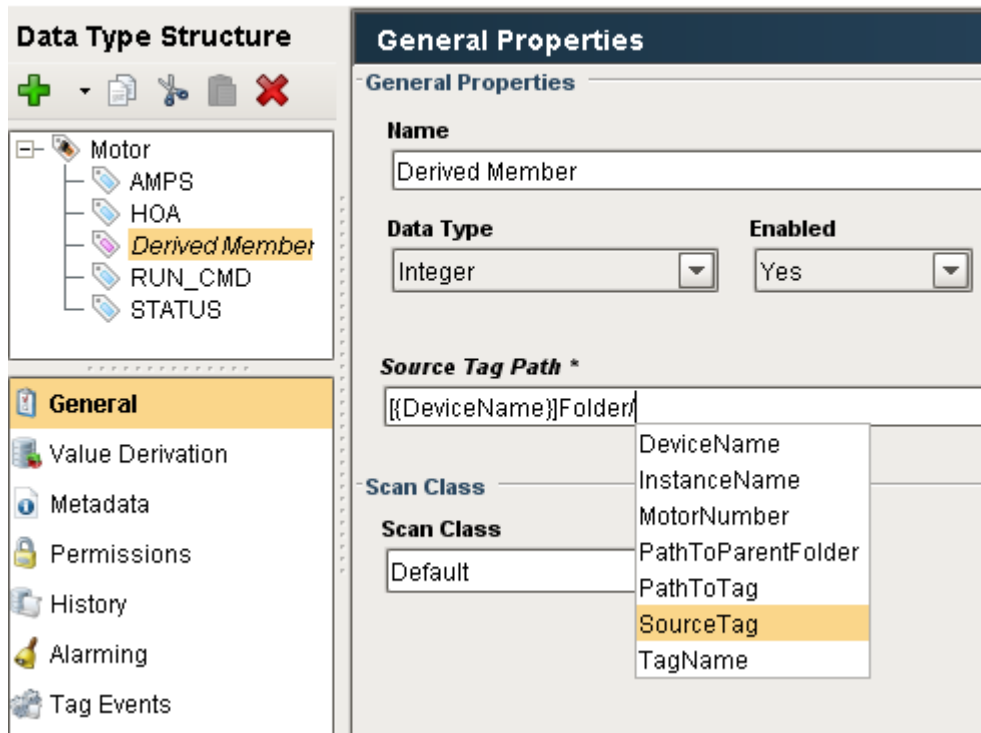
This interface provides an opportunity to scale the written and read value. For example, if the source Tag was in Fahrenheit, and the derived Tag should be Celsius, we could use the following expressions:

```
//Read Expression
({source}-32)*(5/9)

//Write Expression
{value}*(9/5) + 32
```

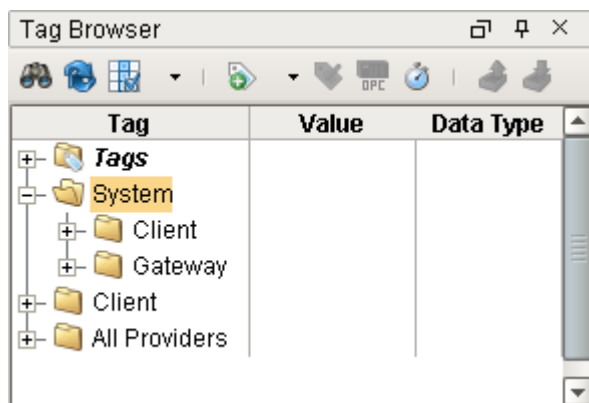
Derived Tags in UDTs

When added as a member of a UDT, Derived Tags may reference peer members. Additionally, UDT parameters may be used in the Source Tag Path.



System Tags

System Tags are status Tags that Ignition provides about the Client and the Gateway. You cannot create any new Tags in the Client or Gateway folders. The Tag names and folder structure are managed by the Ignition Gateway. However, you can modify the Tags in the Gateway folder to perform alarming, history, and scaling. You cannot modify the Client-scoped System Tags.



Client-scoped System Tags

Client-scoped Tags provide status information about the client's system. Every individual client is going to have their own values like IP address, hosting name, username, and more.

Tag	Value	Data Type
Tags		
System		
Client		
Network		
GatewayAddress	http://localhost:...	String
GatewayRedundancyRole	Independent	String
Hostname	jdoe-ws2	String
IPAddress	10.20.6.140	String
System		
User		

Gateway-scoped System Tags

Gateway-scoped Tags provide status information about the Gateway like alarming, database, throughput, memory usage, performance of the server, and much more.

Tag	Value	Data Type
Tags		
System		
Client		
Gateway		
Alarming		
Database		
EAM		
OPC		
Performance		
Available Disk Space (MB)	370,010	Int8
CPU Usage	0	Float8
Disk Utilization	0.2	Float8
Max Memory	2,077,753,344	Int8
Memory Usage	248,173,672	Int8
Memory Utilization	0.12	Float8
Redundancy		
Sessions		
Tags		
CurrentDateTime	2017-01-12 10:55:2...	DateTime
Timezone	America/Los_Angeles	String

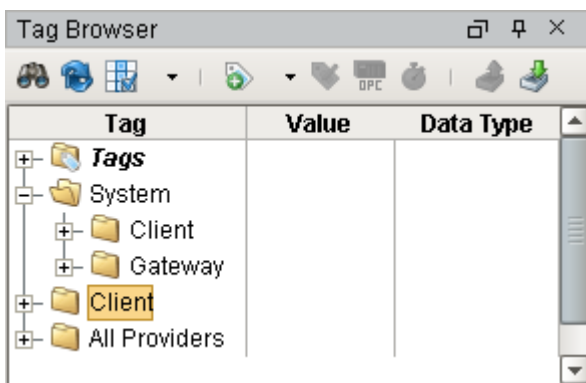
Other System Tags

There are potentially more of these System Tags depending on the modules that you have installed. For example, if the **Mobile Module** is installed you will see a **Mobile** folder under the Client and Gateway folders. This will include information about the Client in addition to the Client folder, but only if the client is opened with the Mobile Module.

Client Tags

Client Tags, as the name implies, are only available for use in Clients. They are scoped at the client level unlike the Gateway Executed Tags. All clients will have the same list of client Tags, however, the actual values are unique and independent for each running Client. In other words, even though client Tags are created in the Designer, each client will create their own instances. This makes them very useful as in-project variables for passing information between screens, and between other parts of the clients, such as scripting.

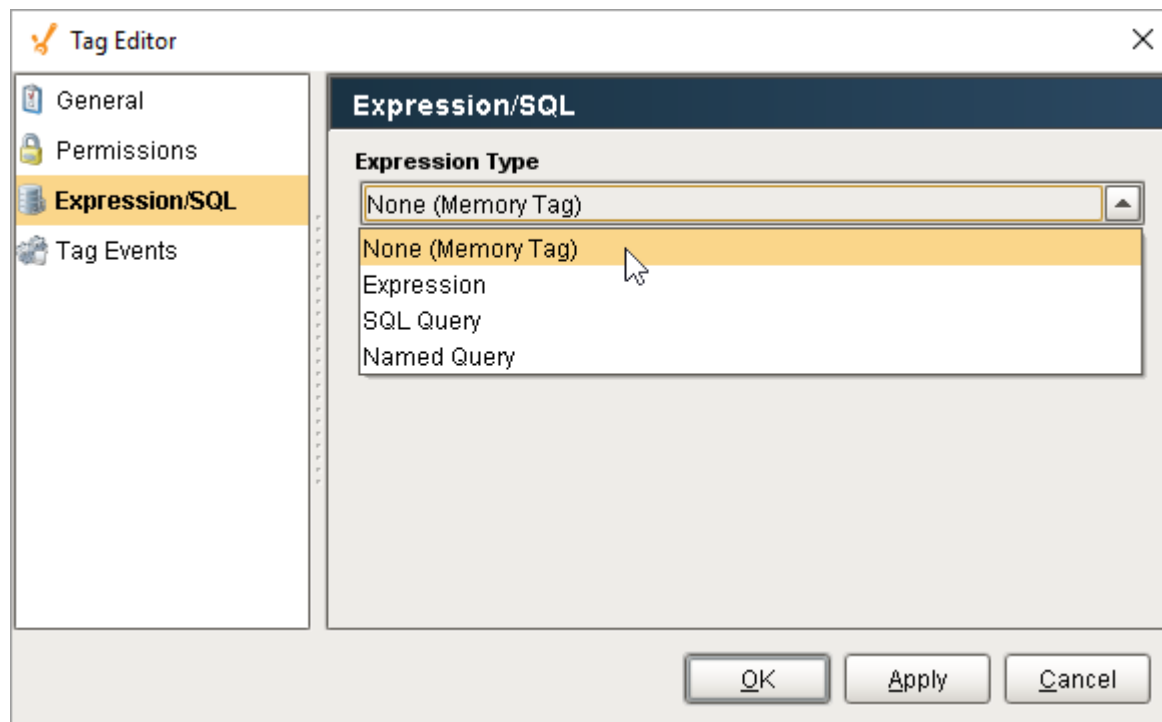
Client Tags support most of the data types that standard Tags do (including datasets), but excluding the array types. Additionally, Client Tags do not have a Scan Class property, so the value will only update when the polling property executes, or a reference in the Client Tag's expression updates.



Expression Type

Client Tags can be configured in one of several ways, based on the **Expression Type** property. The following options are available.

- **None**: Causes the Tag to behave like a **Memory Tag**.
- **Expression**: Allows the Tag to utilize Ignition's Expression Language, much like an **Expression Tag**.
- **Query**: Executes a SQL Query, similar to a **Query Tag**.
- **Named Queries**: **New in 7.9.6** Client Tags may now call **Named Queries**.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Keeping Tags Organized



INDUCTIVE
UNIVERSITY

Keeping Tags Organized

[Watch the video](#)

It is important to give Tags a meaningful structure and arrange them in hierarchical Tag folders so that they are easy to understand, identify, and locate for all developers. You can also rename Tags independent of their Item Path.

Another important concept to consider when naming and organizing your Tags, is to do this early in your project. If you rename or move any of your Tags to another folder, and your Tag is being used in other places, chances are you are going to break the reference to the Tag on your screen. So keeping your Tags organized and defining your Tag structure early on in your project is critical.

Tag Name Rules

The best way to keep your Tags organized is to rename them with something meaningful. By default, Ignition Tags are named after their OPC Server address when a Tag is dragged into the Tag Browser. You can change this name to just about anything that you want. We recommend using names that mean something to your process like 'Motor 3 Amps,' or creating folders in your Tag Browser like 'Motor 3/Amps'. When renaming Tags and folders, there is really only one question to ask: "does this structure make sense?"

When you choose a new name for your Tags and Folders, there are some rules that must be followed. The first character of the Tag name must be one of the following:

- any alphanumeric
- any valid unicode letter
- an underscore

The second character, and every character after that can then be one of the following:

- any alphanumeric
- any valid unicode letter
- an underscore
- a space
- any of the following special characters:
 - ' `'`
 - - `-`
 - : `:`
 - (`(`
 -) `)`

All other special characters can not be used in a Tag name.

Tag Paths

Tags and their properties can be referenced by a string-based path. Each Tag has a unique absolute path and often has many equivalent relative paths when referenced from other Tags. You most often generate these paths by browsing or through dragging. However, it's a good idea to understand how Tag paths work, particularly if you get into [indirect Tag binding](#) or scripting.

A Tag path looks something like this: `[Tag Provider]folder/path/tag.property`

The `folder/path/tag.property` portion of the path may contain the following:

- A Tag
- Any number of nested folders followed by a Tag, separated by forward slashes (/).
- A period (.) followed by a property name after the Tag. Omitting this is equivalent to using the `.Value` property.

The `[Source]` portion surrounded by square braces can have the following options:

Source Option	Meaning	Applicability
[Tag Provider Name]	The name of the Tag provider that hosts the Tag.	OPC, Expression Tags
[] or not specified	The default Tag provider for the current project.	OPC, Expression Tags
[.]	Relative to the folder of the Tag that is being bound.	Expression, Client Tags
[~]	Relative to the Tag provider of the Tag that is being bound (root node).	Expression, Client Tags
[Client]	Refers to a Client Tag.	Client
[System]	Refers to a System Tag.	System

Using Relative Paths

Paths that begin with [.] or [~] are known as relative paths. They are used inside Tags that bind to other Tags, and are relative to the host Tag's path. Using the relative path syntax helps to avoid problems caused by moving Tags and renaming providers.

[.] refers to the Tag's current folder. By using [.] , Tags can be moved from folder to folder without problem (provided that all of the applicable Tags are moved together). Additionally, you can use ".." (two periods) to go back one folder from the current relative position, for example [.]../tag allows you to reference a Tag two folders up.

[~] refers to the Tag's provider root. It can replace the explicit provider name, and thus protect against provider renames and importing/exporting/moving Tags between different providers.

Tag Path Manipulation

Ignition provides a great deal of flexibility for Tag addressing since Tag paths and Tag properties are string-based. The underlying strings that compose a valid Tag path can be assembled from many different parts in with the eventual construction resulting in a valid Tag path.

The following scripting demonstrates this concept. Suppose there was a Tag path to a level indicator in a tank. In this case it is the default Tag provider, Tanks folder, Tank 1 Folder, and the Level Tag.


```
tagPath = "[default]Tanks/Tank 1/Level"
```

But suppose that there was more than just Tank 1 and instead there was Tank 2, Tank 3, Tank 4, etc,. Dynamically changing the Tag paths is simple with Ignition's Tag paths being nothing more than string representations. The following takes the tank number and inserts it into a new Tag path. The tankNumber variable changes the eventual creation of the tagPath.

```
tankNumber = 2  
tagPath = "[default]Tanks/Tank %i/Level" % tankNumber
```

The result of the tagPath variable will be [default]Tanks/Tank 2/Level which is a valid Tag path to the the level sensor for Tank 2.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Tag Quality and Overlays



INDUCTIVE
UNIVERSITY

Tag Quality and Overlays

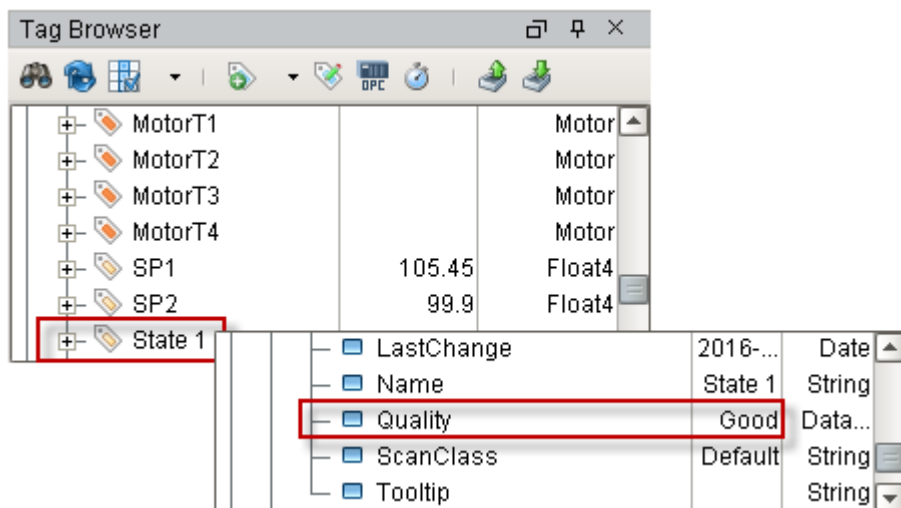
[Watch the video](#)

Tag Quality

Ignition has Quality built into Tags automatically. Data Quality is the measure of how reliable a particular Tag's data is. If a Tag's quality is not Good (192), the value generally should not be trusted, and an overlay is shown to make the Tag's bad quality visible. There are a wide variety of causes of bad data, from network disconnections, to software failure, to invalid Tag configurations.

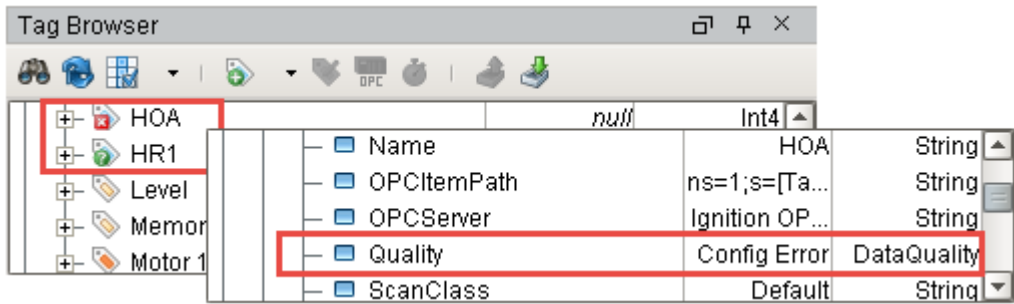
Quality is extremely important and exists on every Tag. It is visible in the Designer and in the Runtime through an Overlay system. There are a few of ways to check Tag Quality.

- In the **Tag Browser**, find your Tag, expand it, and scroll down to the meta property called **Quality**. Here, you can see if the quality of the Tag is either **Good** or **Bad Quality**. This example shows a Good Quality Tag, meaning the Tag can be trusted.

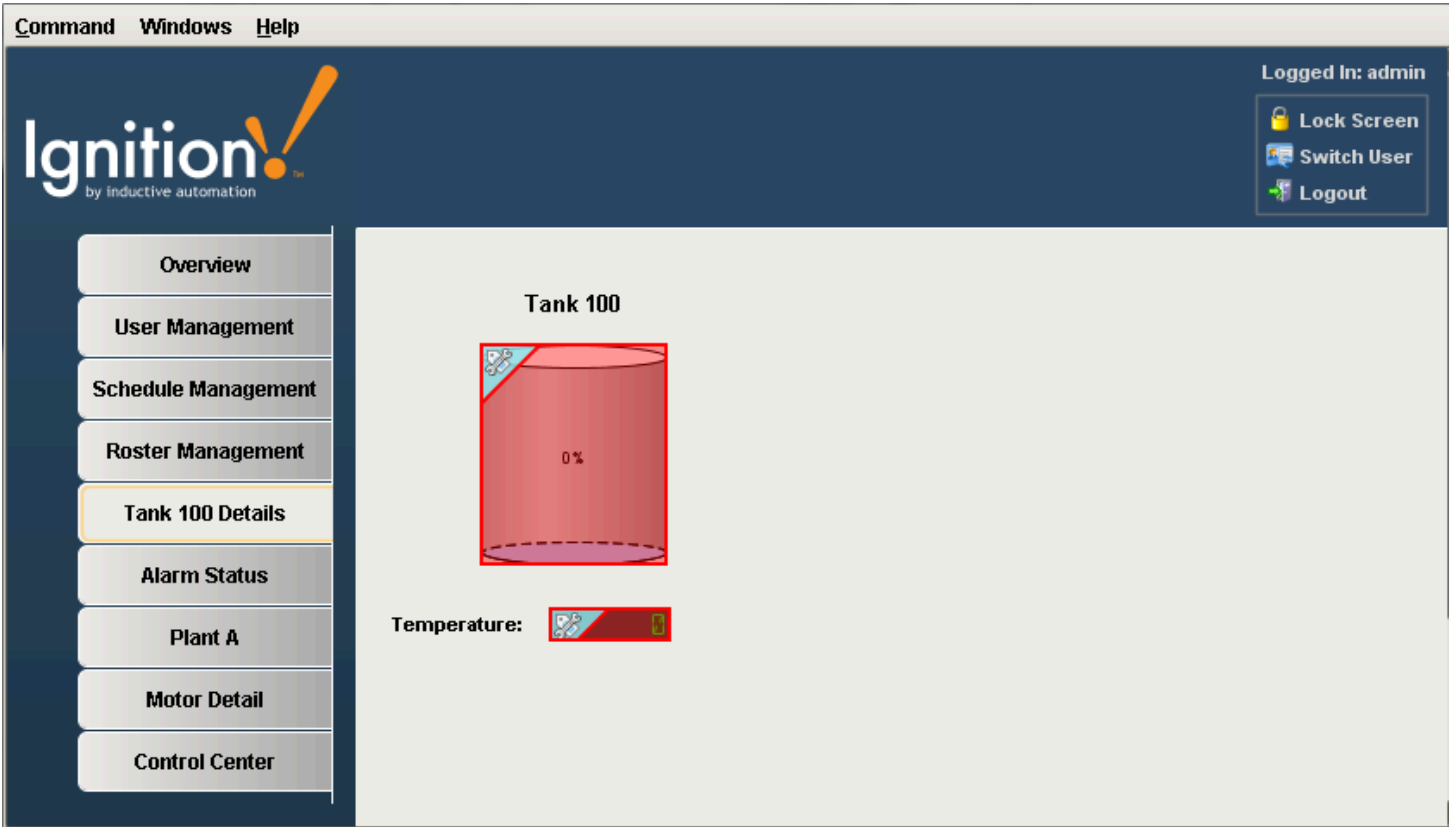


- One obvious indicator if the Tag is of bad quality is if there is an overlay on the Tag itself. This example shows two Tags with overlays that have a quality issue. The HOA Tag was expanded to

show that there is a Configuration Error and the HR1 Tag has an Unknown issue. Ignition typically tries to identify the Quality issue to help you resolve it promptly.



- Quality is also reflected on the component in the window of the Designer and Runtime using an overlay. Using an overlay on a component lets the operator know that they could be looking at a bad value for that Tag. When the overlay goes away and the values start coming in again, the operator knows that it's a valid Tag, and the values can be trusted. Bad Quality overlays on components look like the ones in this example which show a Tag's Evaluation Error.



Primary Data Qualities

The following table outlines the primary data qualities. The most important is **Good**, and that has a value of **192**. There are more values, but these represent the most common:

Quality	Value	Meaning
Good	192	The data has met all criteria for being considered reliable.
OPC_Bad_Data	0	The data is not reliable, further data isn't available.
OPC_Not_Connected	8	The OPC server driving the tag is not currently connected OR a value has not yet been received by the tag from the server.
Config_Error	300	There is a problem with the tag's configuration. The error log may provide more information as to the exact problem.
Comm_Error	301	There is a problem in communication somewhere between the tag and its data source.
Expression_Eval_Error	310	The expression in the tag generated an error during execution. The error log should provide more information on the error.
Good (Provisional)	320	Temporary "Good" value. The value isn't cached because the underlying quality may differ than what appears on the tag.
Tag_Exec_Error	330	There was an error when evaluating the tag.
Type_Conversion_Error	340	The value of the tag could not be converted to the requested data type. Check the assigned data type of the tag.
Driver_Demo_Timeout	900	The system driving the tag is operating in demo mode and has timed out.
Access_Denied	403	The tag permission settings do not allow the current user to view the tag.
Not_Found	404	The tag, or a tag referenced from inside of it, could not be found (incorrect reference path).
Disabled	410	The tag's "enabled" property has been set to false.
Stale	500	The tag has not been evaluated within the expected time frame. There is likely a deeper problem with the tag provider.

Quality	Value	Meaning
GW_Comm_Off	901	When viewing Tags in the Designer, the tags will have this value if communication with the Gateway is turned off from the toolbar.

Tag Quality and Referenced Tags

When Tags reference other Tags, such as in expressions, they will often pass the worst sub-quality up as their own. For example, even though a particular Tag's expression executes without problem, if the expression references a Tag whose quality is "Bad", the expression Tag will also report "Bad."

Component Overlays



INDUCTIVE
UNIVERSITY

Tag and Component Overlays

[Watch the video](#)

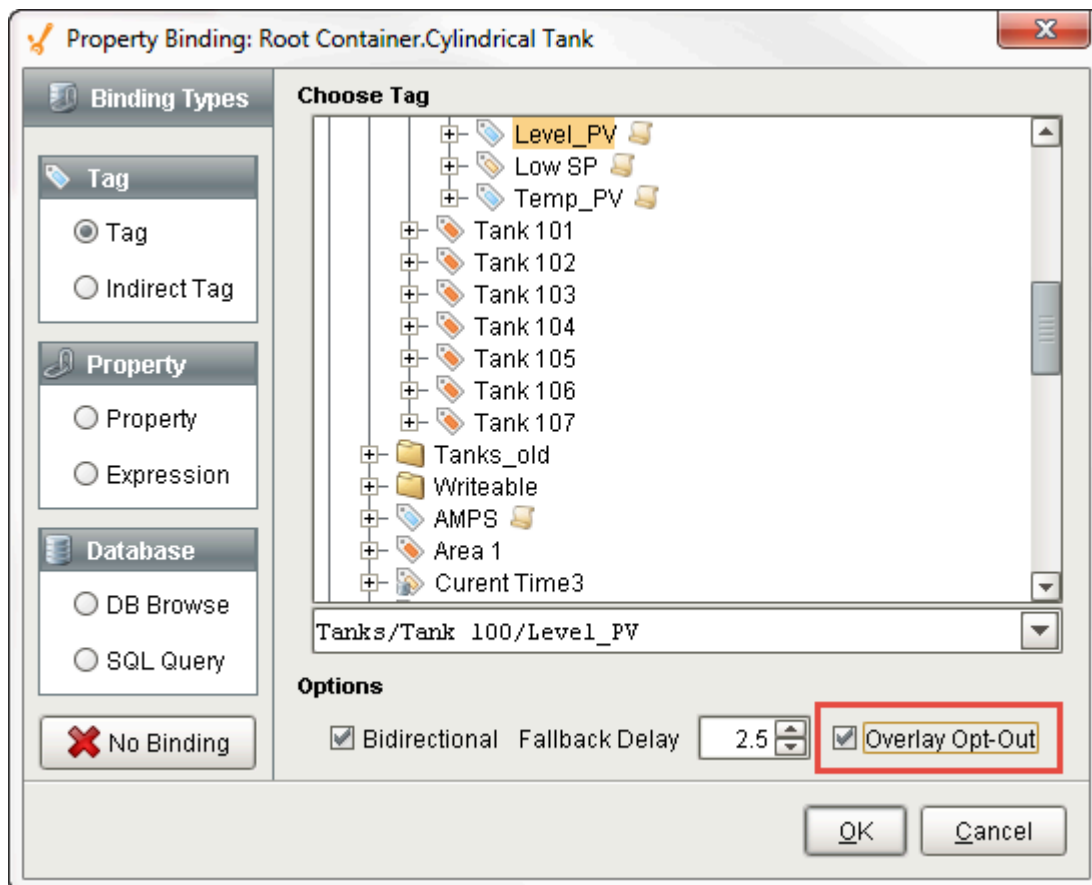
It is especially important in HMIs to be able to gauge the health and accuracy of what is displayed at a glance. In a highly distributed system like Ignition, it is especially important, as the client may be located at quite a distance (maybe across the world) from the physical process it is monitoring and controlling.

For these reasons, components display visual overlays (as shown below) for various reasons to indicate that the data they are displaying is not good, or pending a reply from the device. Each data binding that drives a component is evaluated for quality. If any of these qualities becomes poor, the component will show an overlay. The different overlays can mean different things, denoting their underlying cause. What they indicate is based on the Quality properties of Tags. The table below shows all the available overlays Ignition provides.

Bad Quality OPC_BAD_DATA (0)		Config Error OPC_CONFIG_ERROR (4)	
Not Connected OPC_NOT_CONNECTED (8)		Device Failure OPC_DEVICE_FAILURE (12)	
Sensor Failure OPC_SENSOR_FAILURE (16)		Bad, showing last value OPC_BAD_SHOWING_LAST (20)	
Comm. Failure OPC_COMM_FAIL (24)		Out of Service OPC_OUT_OF_SERVICE (28)	
Waiting OPC_WAITING (32)		Uncertain OPC_UNCERTAIN (64)	
Uncertain, showing last value OPC_UNCERTAIN_SHOWING_LAST (68)		Bad Sensor OPC_SENSOR_BAD (80)	
Limit Exceeded OPC_LIMIT_EXCEEDED (84)		Sub-normal OPC_SUB_NORMAL (88)	
Unknown OPC_UNKNOWN (256)		Config Error CONFIG_ERROR (300)	
Comm. Error COMM_ERROR (301)		Evaluation Error EXPRESSION_EVAL_ERROR (310)	
SQL Error SQL_QUERY_ERROR (311)		DB Connection Error DB_CONN_ERROR (312)	
Tag Evaluation Error TAG_EXEC_ERROR (330)		Type Conversion Error TYPE_CONVERSION_ERROR (340)	
Access Denied ACCESS_DENIED (403)		Not Found NOT_FOUND (404)	
Disabled DISABLED (410)		Stale STALE (500)	
Unknown UNKNOWN (500)		Write Pending WRITE_PENDING (700)	
Demo Expired DEMO_EXPIRED (900)		Gateway Comm. Off GW_COMM_OFF (901)	

Overlay Opt-out

Choosing the Overlay Opt-out option will ignore the quality of the chosen Tag, making it have no effect on the component's quality overlay. The Overlay Op-out option is located in the Tag bindings for components. If this option is enabled, the operator will not see any overlays and will have no indication that the underlying Tag quality is something other than good. A word of caution when you use the Opt-out option because you always want to give the operator some indication that the values they are seeing on the screen can be trusted, and by opting out, you are removing that indicator for the operator.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Tag Scaling Properties



INDUCTIVE
UNIVERSITY

Tag Scaling Properties

[Watch the video](#)

Configuring a Tag's scaling will condition the data for use within the Ignition Designer. Scaling will take the raw PLC value driving a Tag, do some math, and use the resulting value as the value of that Tag. And scaling works both ways! When you write to that Tag, Ignition will scale it in the opposite direction before writing to the PLC.

For example, if the capacity of a tank is 5250 gallons, but the tank's fill level is better represented in the Designer as percentage of 0 through 100, configuring the Tag's scaling property, will result in the Tag displaying 0 through 100, while the actual tank fill moving is between 0 and 5250 gallons. For this example, you can click on the Tag to open the **Tag Editor**, then click on **Numeric**. This allows you to configure the scaling of the Tag. When scaling between a Raw Low and and High, and Scaled Low and High, select the **Linear Scale Mode**. So what Ignition is actually doing, is setting up the calculations behind the scenes to scale the value appropriately.

The screenshot shows the 'Tag Editor' window with the 'Numeric' tab selected. The 'Numeric Options - Scaling & Deadband' dialog box is open. It contains the following settings:

- Scale Mode ***: Linear
- Raw ***: Low 0.0, High 5,250.0
- Scaled ***: Low 0.0, High 100.0
- Clamp Mode**: None
- Deadband**: 1E-4
- Deadband Mode**: Absolute

Buttons at the bottom: OK, Apply, Cancel.

⚠ IF YOU ARE USING SCALING...

The numbers and units don't have to match-up. Scaling is straight from one number to another number. It doesn't matter what the units are, and it doesn't matter what the conversion is. What is important, is that the data type of the Tag must match the data type of the scale value. IE: dividing an integer in the PLC by 10 probably means your Ignition Tag should be a float.

Scaling is not available on Memory Tags: Memory Tags are not driven by an external source such as a PLC or SQL query, so scaling will never be applied. In these scenarios, it is recommended to scale the mechanism that is writing to the Memory Tag instead. Numerical properties of Tags can be scaled allowing automatic bi-directional conversion outside of the PLC. Scaling types include **Linear** scaling, **Square Root** scaling, and **Exponential Filter** scaling. The numerical properties are available to **OPC**, **Expression**, **Database**, and **Client Tags** whose data types are numeric. For a complete list of all of the Tag Scaling properties, see [Tag Properties](#).

Linear Scaling

The value will be scaled linearly between the low and high values, and clamped as appropriate. The linear equation is:

$$\text{ScaledValue} = \Delta S * (\text{Value} - \text{RL}) / \Delta R + \text{SL}$$

Square Root Scaling

The equation for square root scaling is:

$$\text{ScaledValue} = \Delta S * \sqrt{(\text{Value}-\text{RL})/\Delta R} + \text{SL}$$

Where:

- $\Delta S = (\text{ScaledHigh}-\text{ScaledLow})$
- $\Delta R = (\text{RawHigh} - \text{RawLow})$
- $\text{RL} = \text{RawLow}$
- $\text{SL} = \text{ScaledLow}$

Exponential Filter Scaling

This mode implements a simple linear recursive filter to smooth values. The scale factor corresponds to the weight of the smoothing effect, and is a value between 0.0 and 1.0. The smaller the factor, the greater the degree of smoothing.

The equation for the filter is:

$$y(t) = (1-f)*y(t-1)+f*x(t)$$

Where:

- $y(t)$ = the output at time t
- $y(t-1)$ = the previous output
- $x(t)$ = the input value (current value)
- f = the scale factor, with $0.0 \leq f \leq 1.0$

NOTE

Only **good** quality values are considered for the filter. **Bad** quality values are ignored.

Bit Inversion Scaling

This simple scaling mode will the complement of a binary value. If the current value is coming in as 0001_0101 (21), this will return a 1110_1010 (-22) instead. A popular use for this scale mode is that it can be used to invert modbus values if your device stores them in reverse bit order. Note that Bit Inversion Scaling uses a little-endian format.

Examples

Simple Divide by Ten

This is common when storing a single decimal point of precision as an Integer in the PLC. This is to save space by not using a Float type.

- **Raw Low:** 0.0
- **Raw High:** 100.0
- **Scaled Low:** 0.0
- **Scaled High:** 10.0

4-20 Milliamp Signal to Percent

This is common when using a simple pressure sensor. The sensor is calibrated to send 4 milliamps (minimum value) when the tank is empty, and 20 milliamps (maximum value) when the tank is full.

- **Raw Low:** 4.0
- **Raw High:** 20.0
- **Scaled Low:** 0.0
- **Scaled High:** 100.0

4-20 Milliamp Signal to Gallons

This is common when using a simple pressure sensor. The sensor is calibrated to send 4 milliamps (minimum value) when the tank is empty, and 20 milliamps (maximum value) when the tank is full (5000 gallons).

NOTE

There is no direct conversion between amps and gallons. In scaling it doesn't matter.

- **Raw Low:** 4.0
- **Raw High:** 20.0
- **Scaled Low:** 0.0
- **Scaled High:** 5000.0

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Importing and Exporting Tags



INDUCTIVE
UNIVERSITY

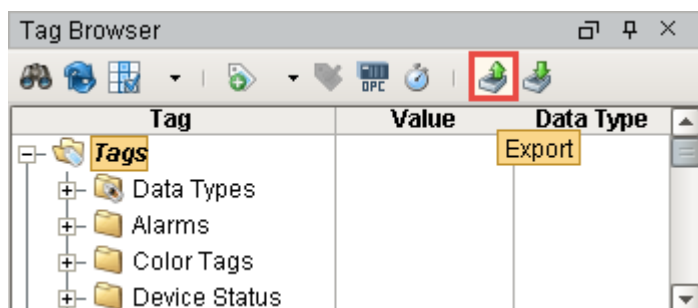
Importing and Exporting Tags

[Watch the video](#)

Ignition can export and import Tag configurations to and from **XML** (Extensible Markup Language) or **CSV** (Comma Separated Value) file formats. This allows you to make changes to the Tag structure outside the Designer. The importing and exporting feature also becomes really important if you have a Tag database that you exported from a legacy system that you want to import into an Ignition database.

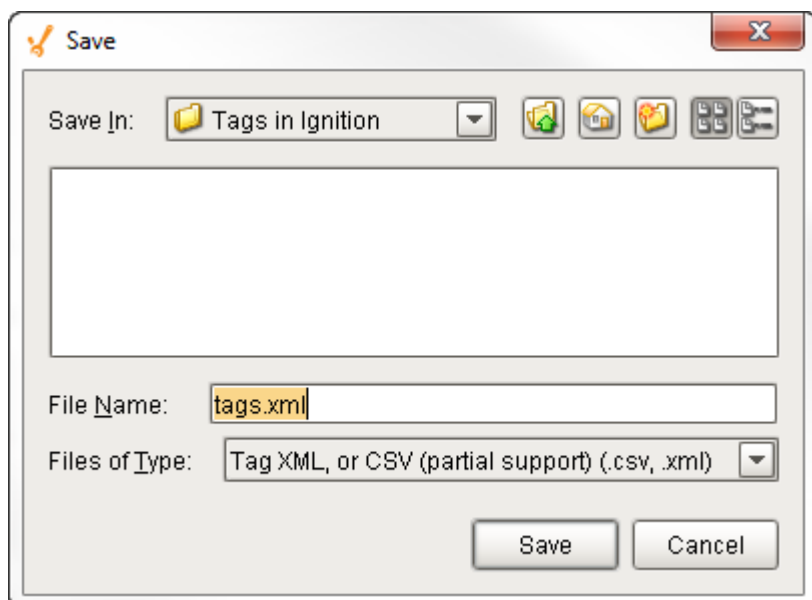
To Import and Export Tags

1. Go to the **Designer**, select the desired folder from the **Tag Browser** that contains the Tags that you want to export. If you want to select all your Tags, click on the Tags folder. If you want to click on one Tag, click on the individual Tag.
2. On the **Tag Browser** toolbar, click the **Export** icon to export, or **Import** icon to load a previously exported file.



3. Specify the folder you want to export/input your Tags either to or from. Then choose either the **.xml** or **.csv** file type to export or import your Tags. The **Save** (to export) or the **Open** (to import) window is displayed.

Importing Tags from XML will overwrite Tags if an **exact** match is found.



Tag Export Localization

New in 7.9.13

When exporting tags, we now include the designer's system locale and format numbers in the output according to that locale. When importing back in, the locale in the import file is honored and values interpreted accordingly. If a locale is not specified, the designer's system locale is used.

Dates are explicitly excluded from locale-based formatting.

Formats

XML Format

Extensible Markup Language (XML) is a markup language that defines a set of rules to encode documents. XML is easy for software to read and export, therefore, exporting Tags to an XML format results in simple and reliable transport of all Tag configuration settings. The XML format is best for cases where you are interested in transferring Tag definitions and configurations from one Gateway to another. However, it is not optimized for directly editing its contents. Modern variations of Microsoft Excel can open and display the contents of an XML file. You can also open the contents of the XML file in Notepad or another text editor, edit the contents, and import the XML file back into Ignition.

CSV Format

The CSV format is often used to export Tags with the intent of making mass edits to the Tag definitions and configurations in a spreadsheet program such as Microsoft Excel. The Tags can be

edited and imported back into the Gateway. Alarm properties are not included in CSV export format, but are included in XML export format.

 INFO

There are many configuration settings for Tags than what is displayed in a CSV or XML export file. The Tag export feature only exports the configuration properties that have been edited in at least one of the Tags in the selected export folder. Therefore, to ensure the desired configuration setting is available in the export file, at least one Tag within the selected export folder must have that configuration property changed.

Property Values in XML or CSV Export Files

When viewing a CSV file which has been exported from Ignition, you will see many columns each showing a configuration property value. The property consists of its name and its value, in some cases there might be additional attributes available.

The following table shows the configuration property names and values you will see in an exported CSV file.

Tag Properties

Property Name	Type	Values (if applicable)	Description
Value			The value of the tag, dependent on the data type.
Data Type	Int	0 - Int1 1 - Int2 2 - Int4 3 - Int8 4 - Float4 5 - Float8 6 - Boolean 7 - String 8 - DateTime 9 - DataSet	
Enabled	Boolean	true/false	
Tagtype	Int	0 - OPC Tag 1 - DB Tag (see ExpressionType) 2 - Client Tag 6 - Folder 13 - Derived Tag	Determines the type of the tag. A value of one is a "DB Tag", which is either a Memory tag, Query tag, or Expression tag, depending on the value of the ExpressionType field.
ExpressionType	Int	0 - None 1 - Expression 2 - SQL Query	Used in conjunction when the TagType is set to 1 (DB Tag). Otherwise this field is ignored.
AccessRights	Int	0 - Read Only 1 - Read/Write 2 - Custom	If custom, will be defined by a Permissions Tag.
OPCServer	String		
OPCItemPath	String		

Property Name	Type	Values (if applicable)	Description
OPCWriteBackServer	String		Write back target for expression Tags.
OPCWriteBackItemPath	String		
ScaleMode	Int	0 - Off 1 - Linear 2 - Square Root 3 - Exponential Filter	
ScaleFactor	Float		For exponential filter
RawLow	Float		Defines scale range
RawHigh	Float		
ScaledLow	Float		
ScaledHigh	Float		
ClampMode	Int	0 - None 1 - Low 2 - High 3 - Both	
Deadband	Float		
DeadbandMode	Int	0 - Absolute 1 - Percentage	
FormatString	String		
EngUnit	String		
EngLow	Float		

Property Name	Type	Values (if applicable)	Description
EngHigh	Float		
EngLimitMode	Int	0 - None 1 - Low 2 - High 3 - Both	
Tooltip	String		
Documentation	String		
DriverName	String		Used for external tags
ScanClass	String		The export will only include the name of the Scanclass, not the configuration of the Scanclass itself. A Scanclass with the same name needs to already exist on the Gateway that the Tags are being imported to, prior to importing them.
HistoryEnabled	Boolean	true/false	
PrimaryHistoryProvider	String		The history provider to use if storing history
HistoricalDeadband	Float		
HistoricalDeadbandMode	Int	0 - Absolute 1 - Percentage	
HistoricalScanclass	String		

Property Name	Type	Values (if applicable)	Description
InterpolationMode	Int	0 - Discrete 2 - Analog (deadband) 3 - Analog (compressed)	How values are interpolated. 2 exists for backwards compatibility (and is equivalent to 1), but only 0 or 3 should be used in the future.
HistoryTimestampSource	Int	0 - System 1 - Value	
HistoryMaxAgeMode	Int	0 - Unlimited 1 - Limited	
HistoryMaxAge	Int		Max cycles between storage.
UDTParentType	String		The path to the parent UDT type. Used by sub-types and instances.

Alarm Property Values in XML Export Files

When viewing an XML file which has been exported from Ignition, you will see the above fields as well as extra fields for any configured alarms.

The following table shows the alarm property names and possible values you will see in an exported XML file.

Alarm Properties

Property Name	Type	Values (if applicable)	Description
Enabled	Boolean	true/false	
Priority	String	Diagnostic, Low, Medium, High, Critical	May also be numeric, 0-4.
DisplayPath	String		
ActivePipeline	String		
ClearPipeline			
Deadband	Float		
DeadbandEvalMode	Integer	0 - Absolute 1 - Percentage	
TimeOnDelaySeconds	Float		
TimeOffDelaySeconds	Float		
TimestampSource	Int	0 - System 1 - Value	
AckMode	Int	0 - Unused 1 - Auto 2 - Manual	
Notes	String		
AckNotesReqd	Boolean	true/false	
ShelvingAllowed	Boolean	true/false	
Mode	String	Equality Inequality AboveValue BelowValue	

Property Name	Type	Values (if applicable)	Description
		BetweenValues OutsideValues OutOfEngRange BadQuality AnyChange Bit OnCondition	
SetpointA			The setpoint, or the low setpoint for dual value modes.
SetpointB			The high setpoint for dual setpoint modes.
InclusiveA	Boolean		
InclusiveB	Boolean		
BitOnZero	Boolean		Used by the Bit condition, to indicate that 0 is active .
BitPosition			Bit to use in Bit condition.
ActiveCondition	Boolean		The property that drives the OnCondition mode.
AnyChange	Boolean		Whether or not to apply the any change behavior in applicable modes.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Array and Dataset Tags

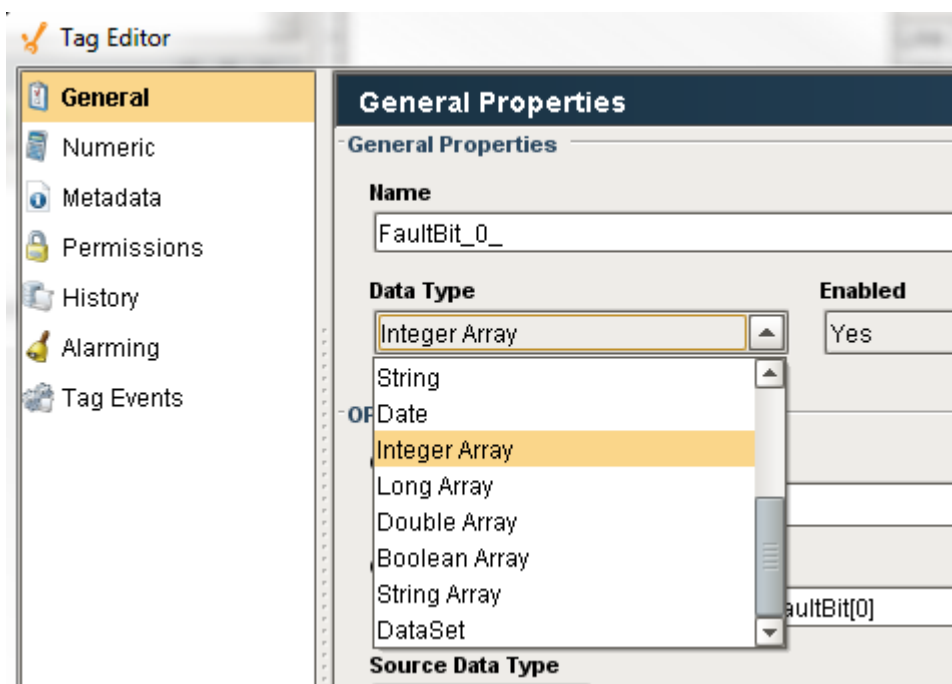


INDUCTIVE
UNIVERSITY

Array and Dataset Tags

[Watch the video](#)

Array and Dataset data types are now available on Tags. Both data types allow for multiple data points to be stored in a single Tag. Configuring a Tag as an array or dataset is as easy as changing the data type in the Tag Editor.



Array Tags

Many OPC servers and drivers already support array type Tags, and now each element in the array can easily be represented with the array data types in Ignition. Additionally, array data types can be used with devices that do not support array types, and will instead expose each bit in the value.

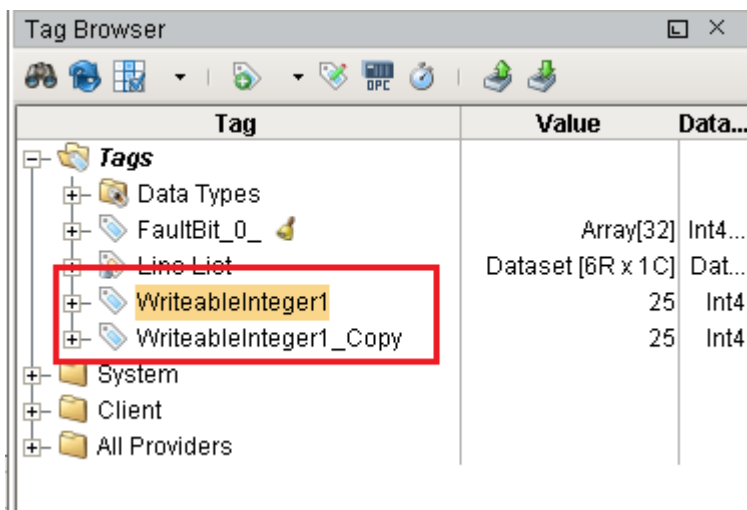
Array Tag Write-Back

New in 7.9.6 OPC Array Tags now support writing back to the device. How this is done can vary, depending on the type of OPC Server in use. Some OPC Servers support writes to individual array elements, where a write would occur just like any other Tag write. However, some OPC Servers do not support individual element writes, which means the whole array will need to be written back to the array tag, even if only a single element is changing.

Array Alarm Example

Because the core data type of each element in the array is the same, it is possible to apply Tag History, Alarming, or Scaling configurations onto the array, and these configurations will be inherited by each element.

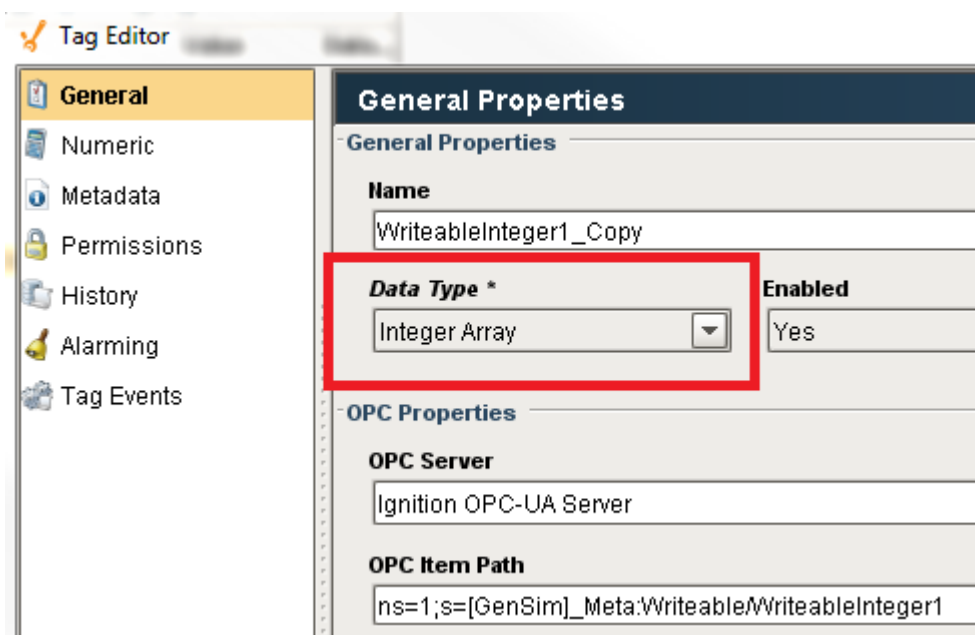
1. The WriteableInteger1 and WriteableInteger1_Copy Tags below are using the same OPC Item Path, and both are currently configured as Integer Tags.



Tag	Value	Data...
Tags		
Data Types		
FaultBit_0_		
Line List		
WriteableInteger1		
WriteableInteger1_Copy		
System		
Client		
All Providers		

Tag	Value	Data...
Array[32]		Int4...
Dataset [6R x 1C]		Dat...
WriteableInteger1	25	Int4
WriteableInteger1_Copy	25	Int4

2. Edit the WriteableInteger1_Copy Tag, and change its data type to an Integer Array, and click **OK**.



Tag Editor

General

General Properties

Name
WriteableInteger1_Copy

Data Type *
Integer Array

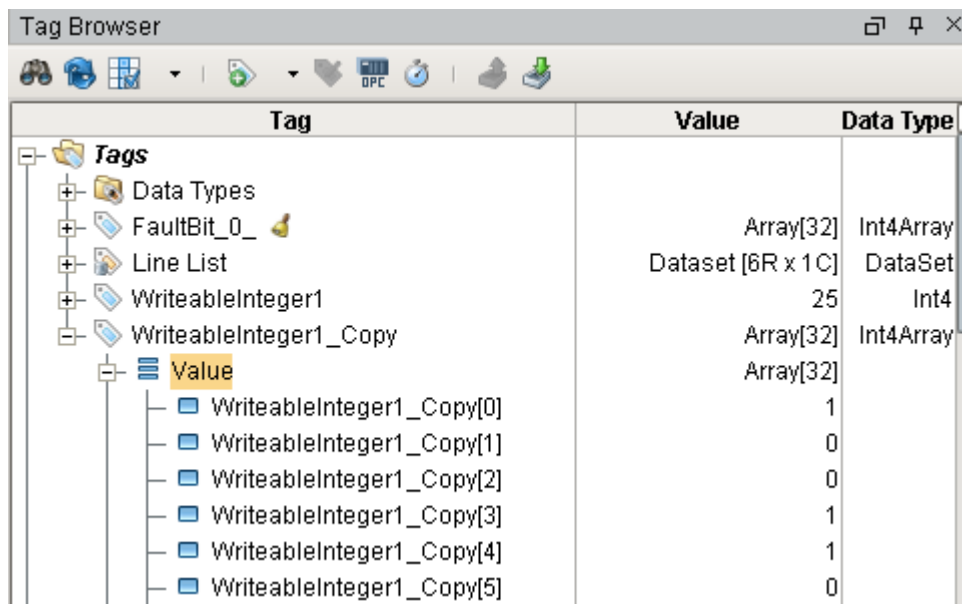
Enabled
Yes

OPC Properties

OPC Server
Ignition OPC-UA Server

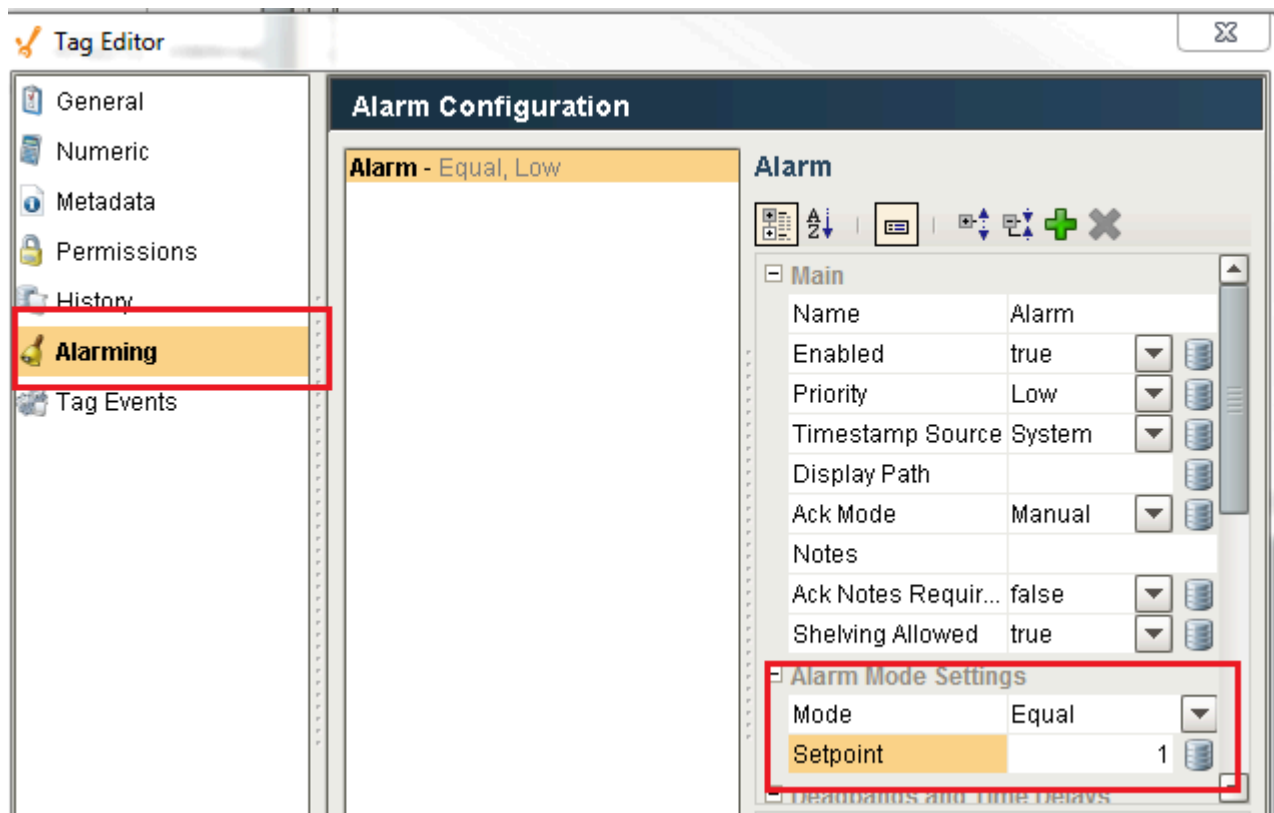
OPC Item Path
ns=1;s=[GenSim]_Meta:Writeable/WriteableInteger1

3. Once the Tag finishes re-subscribing, the value for WriteableInteger1_Copy will now show "Array[32]" instead of the previous value. Expanding the Tag reveals a new Value section, and expanding this section will show the value of each bit.



Tag	Value	Data Type
Tags		
Data Types		
FaultBit_0_	Array[32]	Int4Array
Line List	Dataset [6R x 1C]	DataSet
WriteableInteger1	25	Int4
WriteableInteger1_Copy	Array[32]	Int4Array
Value	Array[32]	
WriteableInteger1_Copy[0]	1	
WriteableInteger1_Copy[1]	0	
WriteableInteger1_Copy[2]	0	
WriteableInteger1_Copy[3]	1	
WriteableInteger1_Copy[4]	1	
WriteableInteger1_Copy[5]	0	

4. Next, edit the WriteableInteger1_Copy Tag again, and head to the Alarming section. Create a new alarm with Mode set to **Equal**, but change the Setpoint to "1," and then click **OK**. This way an alarm will be active for each element that is set to 1.



Tag Editor

Alarm Configuration

Alarm - Equal, Low

Alarm

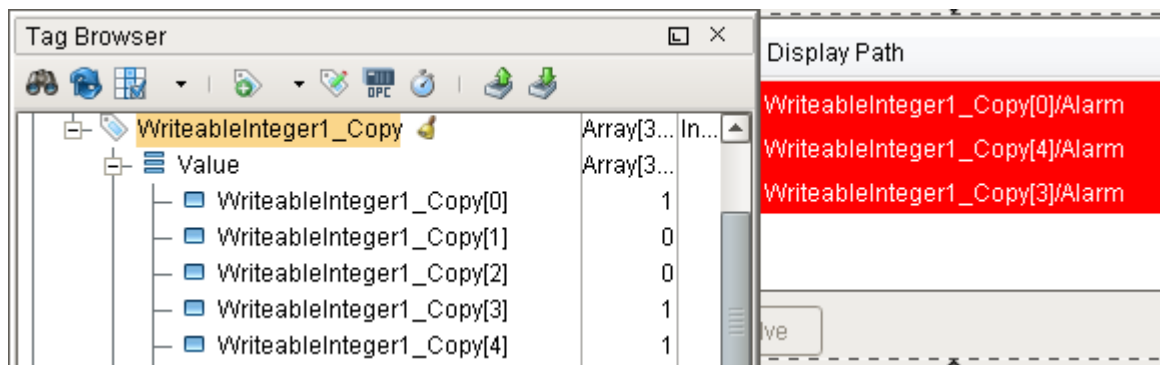
Main

Name	Alarm
Enabled	true
Priority	Low
Timestamp Source	System
Display Path	
Ack Mode	Manual
Notes	
Ack Notes Require...	false
Shelving Allowed	true

Alarm Mode Settings

Mode	Equal
Setpoint	1

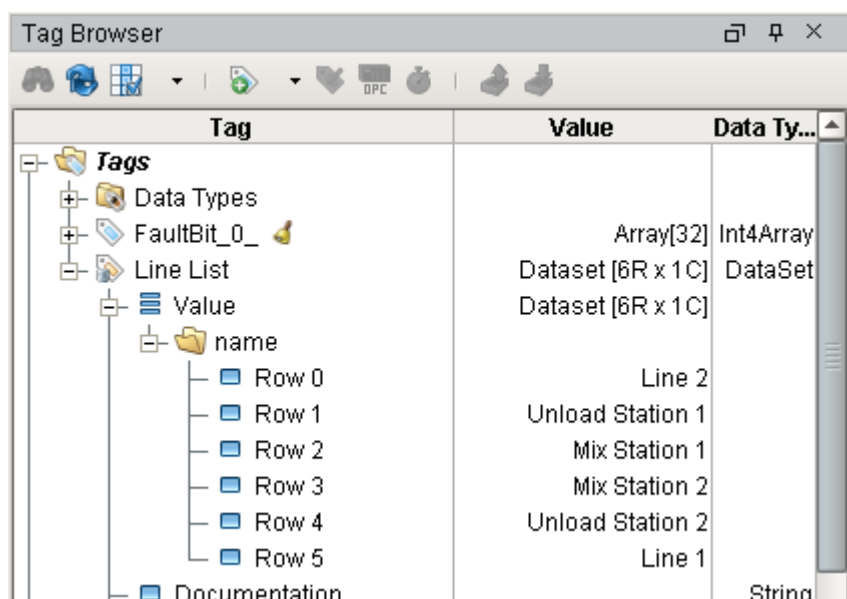
5. Create a new Alarm Status Table report, and you should see an alarm for each true element.



Dataset Tags

Dataset Tags allow multiple rows and columns worth of data to be stored in a Tag. Each column is exposed as a separate folder in the Tag (i.e., the "name" folder in the image below). Dataset Tags can be driven by a query, so it's possible to query for multiple columns on a row in a single Tag. This is much more efficient than using multiple query Tags (and thus multiple queries) to retrieve the same data.

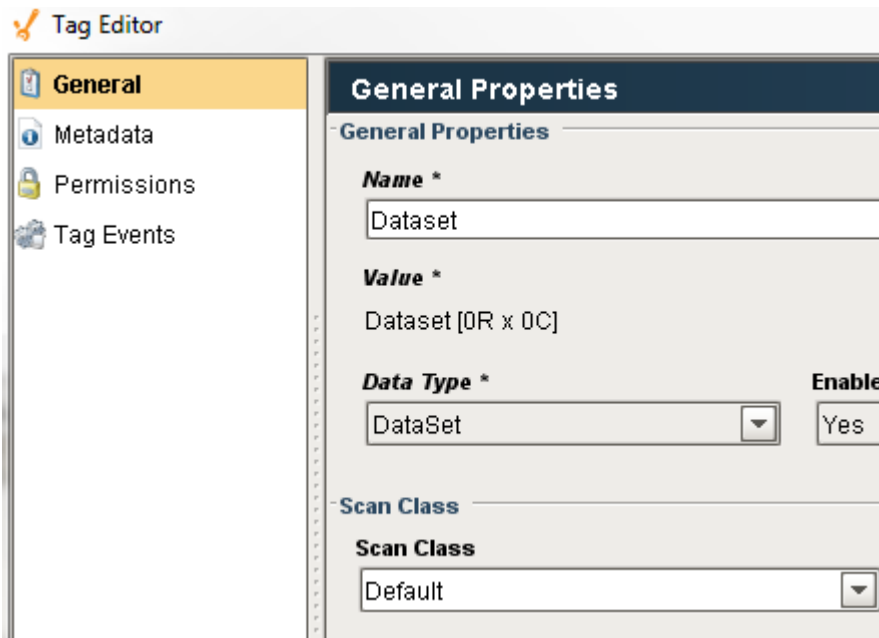
Memory Tags that use the Dataset type support bi-directional bindings in tables.



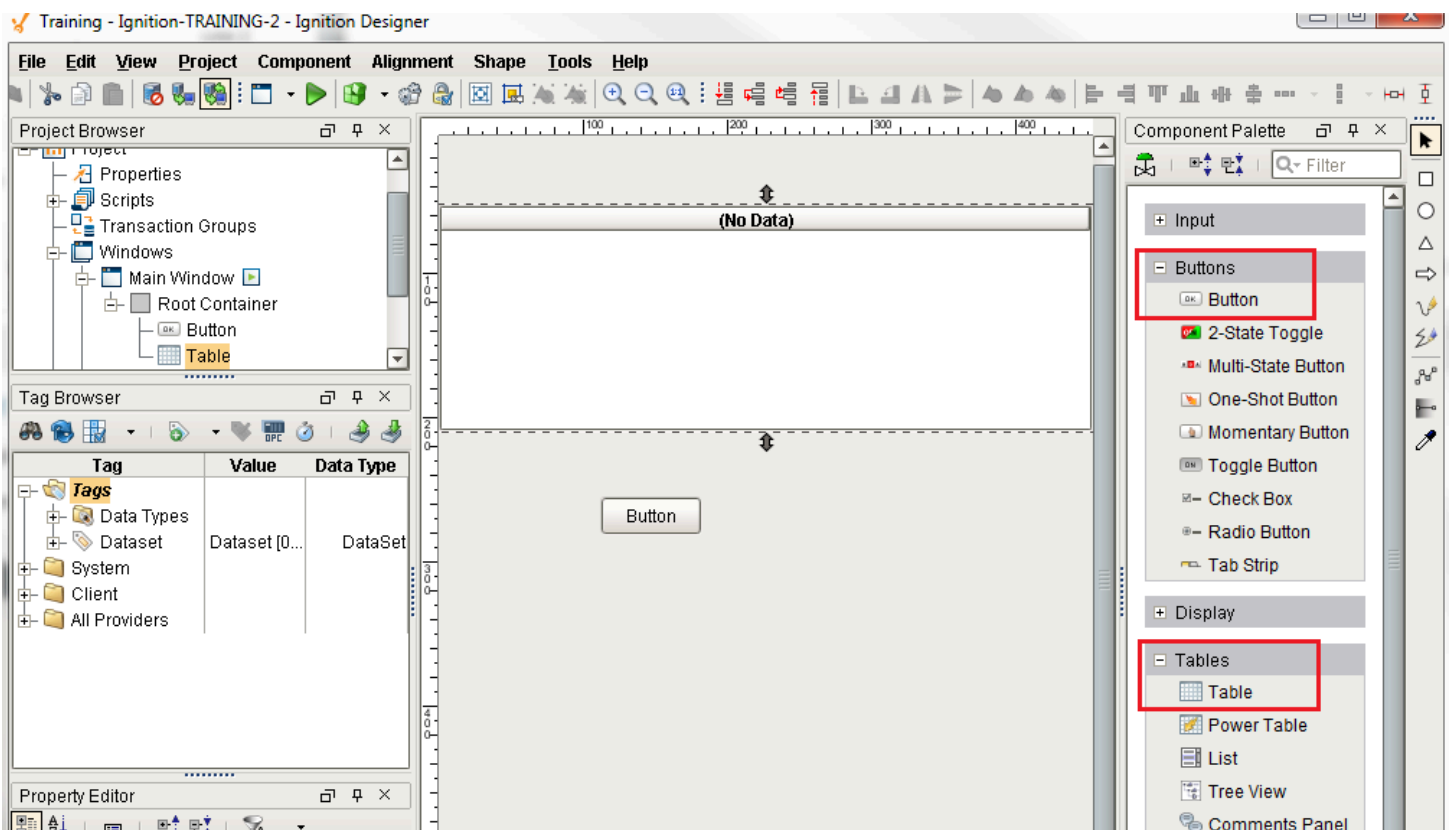
Dataset Tag Write Example

The following example will write the contents of a Table component to a Dataset Tag.

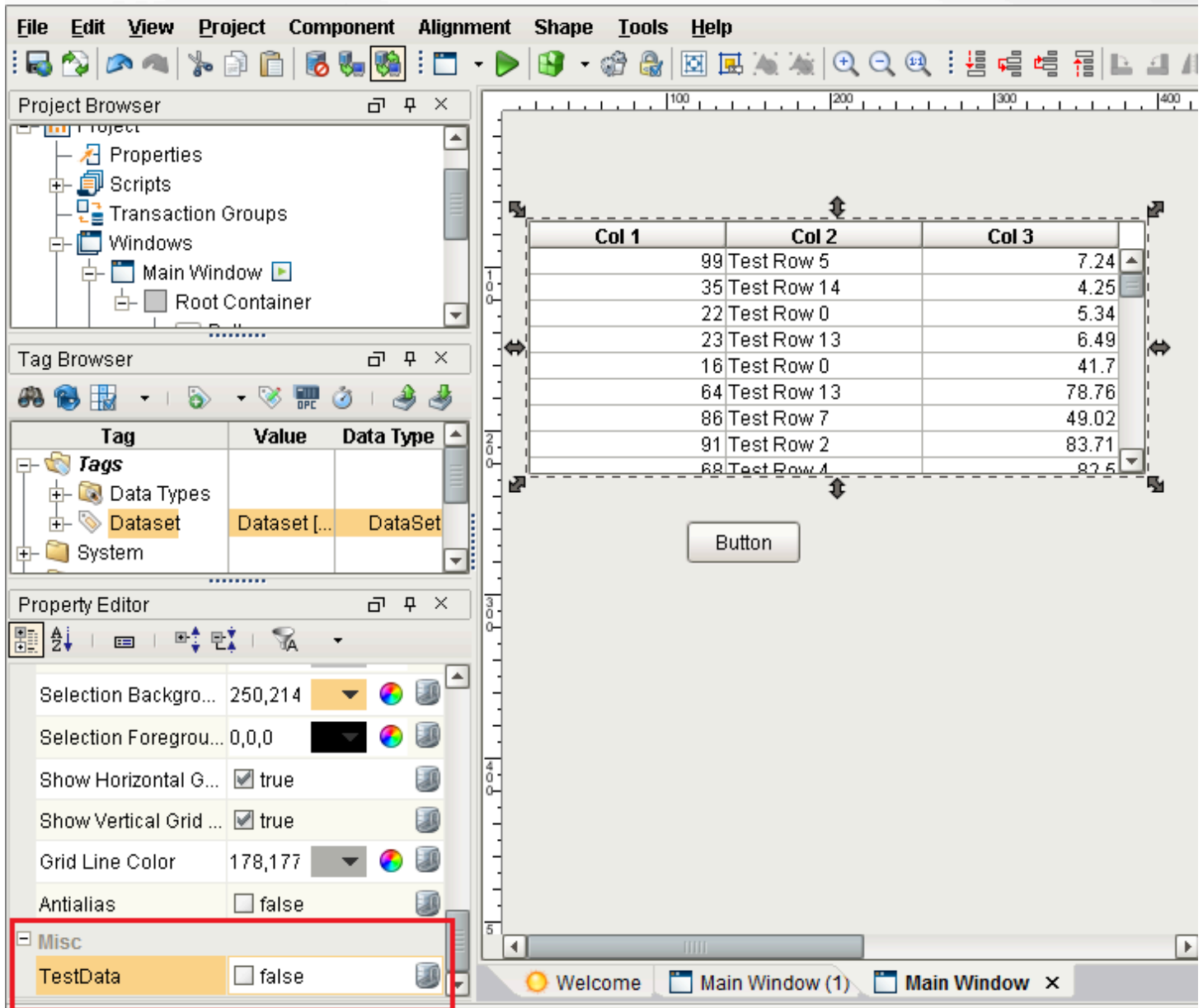
1. Create a new **Memory Tag**. Name it **Dataset**, and change the data type to **Dataset**. The Dataset will be empty by default.



2. On a new window, add a **Button** component and **Table** component.



3. Generate some test data on the Table by toggling the **TestData** property.

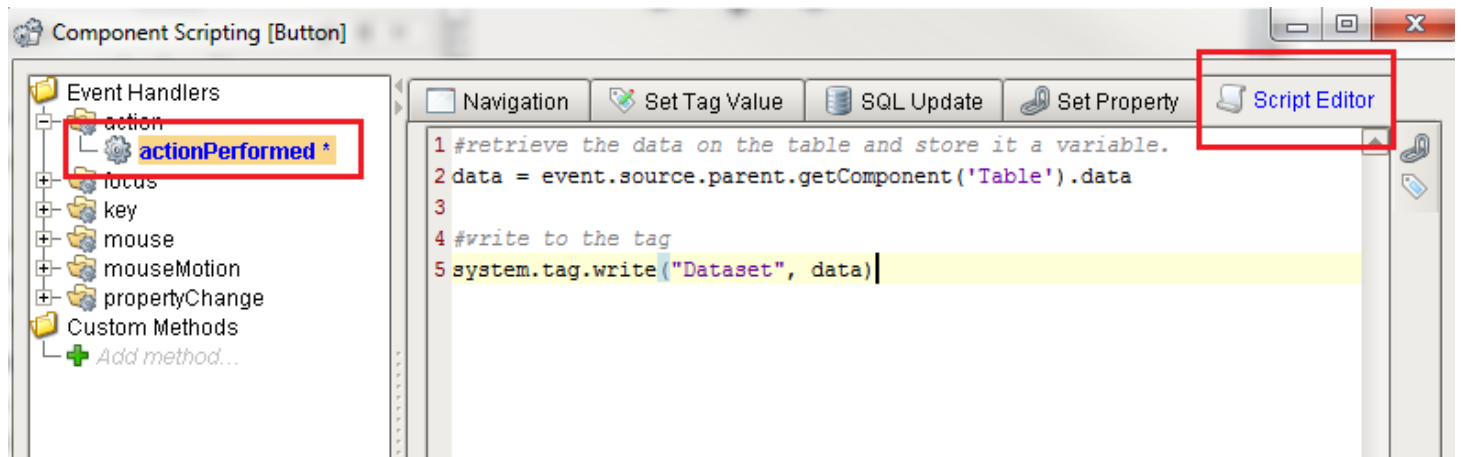


- Next, the **Button** component will need a script to handle writing to the Tag. Right click on the Button component, and select **Scripting**. Choose the **actionPerformed Event Handler**, and click on the **Script Editor** tab. Use the following code:

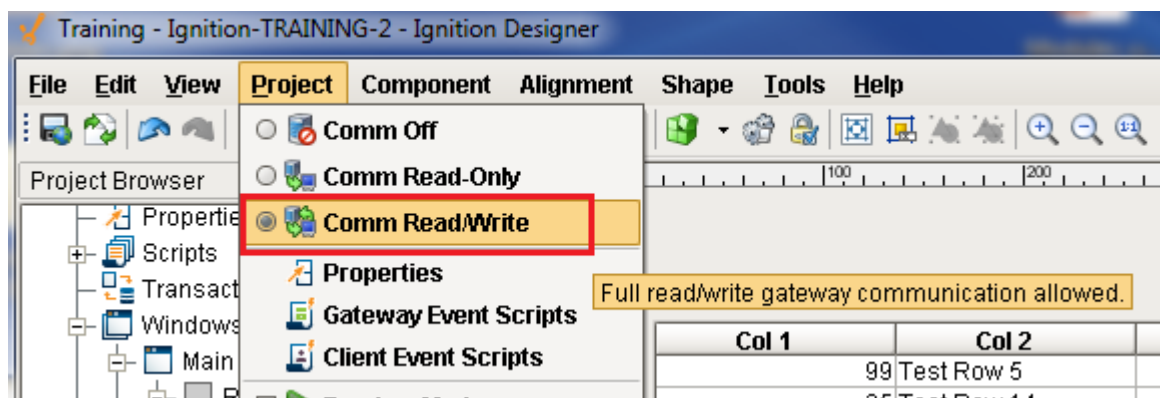
Code Snippet

```
#retrieve the data on the table and store it as a variable.
data = event.source.parent.getComponent('Table').data

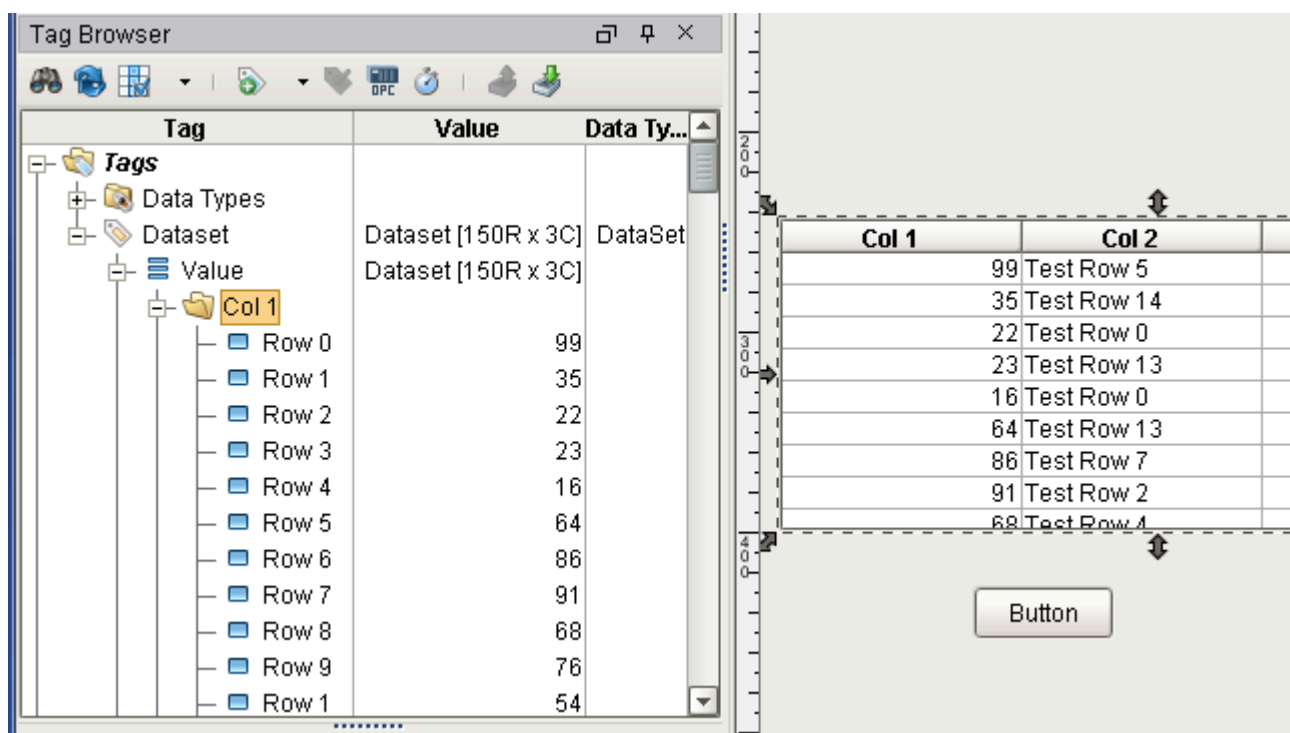
#write to the tag
system.tag.write("Dataset", data)
```



5. Make sure the Designer is set to **Read/Write Mode**: **Project menu > Comm Read/Write**.



6. Put the Designer into **Preview Mode**, and click on the **Button** component. The data in the table has been passed to the Tag. Expanding the Tag in the Tag Browser will show the new values on the Tag.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

External Tag Provider Reference

Important

If you are using the default Tag provider, none of this is applicable. The following information is only valid for new Tag providers that you create using the **External Tag Provider** type.

The information provided here requires an understanding of Tags and how they work. It is an advanced reference to how the tables of external Tags providers are structured, and an overview of the concepts of Tag execution. If you are a new user, it is suggested that you read the [Understanding Tags](#) section first.

SQLT Table Descriptions

Basic Concepts and Data Flow

Tags operate through tables created in the database.

Tag Configuration Table Names and Descriptions

Name	Description
sqlt_core	The core Tag information table, has one entry per Tag. Defines fundamental properties like data type, as well as the current value of the Tag. Is monitored by the provider to determine value and configuration changes.
sqlt_meta	Provides additional properties for Tags. Only consulted when Tag configuration has changed.
sqlt_as	Provides alert state configuration for Tags which utilize alerting.
sqlt_perm	Provides custom permission settings for Tags set to use them.

Operations Table Names and Descriptions

Name	Description
sqlt_sc	Contains the definitions of scan classes, which dictate how Tags are executed.
sqlt_sci	Contains an entry for each scan class from sqlt_sc, for each driver currently driving Tags. Used to verify that drivers are properly executing.
sqlt_drv	Contains an entry for each Tags driver. Only really used for browsing Tags.
sqlt_err	Contains errors that have occurred executing Tags.
sqlt_wq	The "write queue". All write requests are entered into this table, where the driver will detect and execute them. The result will be written back by the driver, and will be noticed by the provider.

Tag Execution Concepts

- **Polling** – Many operations require polling of the database by either the driver or the provider. To ensure efficiency, all polling operations utilize indexed timestamp fields. This allows the database to do very little work when nothing has changed.
- **Tag Configuration** – Tags are configured by inserting or modifying the appropriate entries in the configuration tables above. Configuration change is signaled to the provider by updating the **configchange** of **sqlt_core** to be the current time. Deleting a Tag works by setting its **deleted** column and then **touching** config change. This will inform all drivers and providers to remove the Tag from memory. At some point later, a daemon will delete the Tag information from the database.
- **Tag Execution, drivers** – Each Tag has a **drivername** property that indicates which driver is responsible for executing it. Other drivers and providers with different names will treat the Tag as an **external** Tag – a Tag driven by a different entity – and will only monitor its value.
- **Tag Execution, scan classes** – Each Tag is assigned to a scan class. The idea is that scan classes will define how often the Tag should execute, as well as provide more advanced options like leased and driven execution. In reality, the Tag driver is free to execute Tags as it desires, but it is important to understand how the scan classes and the **sqlt_sci** table are expected to work, as that is how the provider will verify that the Tags are being executed.

- **Tag Monitoring** – Both providers and drivers generally monitor Tag value and configuration changes. In general, the entities will monitor Tags whose **drivername** isn't equal to their own, which for providers means all Tags, since providers don't have a driver name. Monitoring occurs by selecting the Tag values (or any information desired) from the appropriate table where one of the indexed timestamp columns is greater than the last checked time. The provider/driver will then store that time in memory as the last check, and will use it in the next poll.

Table Reference

The following is a reference list for the table structures of all the tables listed above. In general, all integer time values are in milliseconds.

Column	Data Type	Notes
id	integer	Auto-incrementing, unique id for the Tag
name	string	Name of Tag
path	string	Folder path, in form of path/to/
drivename	string	Name of driver responsible for executing Tags
tagtype	integer / TagType enum	The type of Tag - that is, OPC, DB, and so on
datatype	integer / DataType enum	The type of data provided by the Tag
enabled	integer (0 or 1)	Whether the Tag is enabled for execution
accessrights	integer / AccessRightsenum	Access permissions for the Tag
scanclass	integer	ID of the scan class for the Tag
intvalue	integer	Value column used if Tag has integer data
floatvalue	double	Value column for float/real data
stringvalue	string	Value column for string data
datevalue	datetime	Value column for date data
dataintegrity	integer / DataQualityenum	Current quality of the value
deleted	integer (0 or 1)	Whether the Tag is deleted or not
valuechange	datetime	The last time that the value changed
configchange	datetime	The last time that the Tag's config changed

Column	Data Type	Notes
id	integer	Auto-incrementing unique id
name	string	Name of the scan class
lorate	integer	The slower rate to run at, in milliseconds. Only rate used if scan class mode is direct
hirate	integer	Higher rate, in ms. Only used if scan class is driver or leased
drivingtagpath	string	Path to Tag to watch if mode is driven
comparison	integer / Comparison enum	Operation to apply to driving Tag in driven mode
comparevalue	double	Value to compare driving Tag to for driven mode
mode	integer / Scan class mode enum	The mode of the scan class
staletimeout	integer	Time, in milliseconds, before scan class is determined to not be running
leaseexpire	datetime	The time that the lease should expire, if using leased mode
configchange	datetime	The last time that the scan class has been modified
deleted	integer (0 or 1)	Whether the scan class has been deleted

sqlt_meta

Column	Data Type	Notes
tagid	Integer	ID of tag that the property belongs to
name	string	The well-known property name
intval	integer	Value, if property has integer type
floatval	double	Value if property has float type
stringval	string	Value, if property has string type

Column	Data Type	Notes
sc_id	integer	The id of the scan class represented
drivername	string	The driver executing this instance
lastexec	datetime	Last time that the scan class executed
lastexecrate	integer	The rate of the scan class at last execution
lastecexduration	integer	Time, in ms, that the scan class took to execute
lastexecopcwrite	integer	Writes to OPC performed during last execution
lastexecopcreads	integer	Value updates from OPC processed in last execution
lastexecdbwrites	integer	Writes to DB performed during last execution
lastexecdbreads	integer	Value updates from the database processed during the last execution
lastecexdelay	integer	The delay between when the scan class should have ran and when it actually ran for the last execution
avgexecduration	integer	The average duration time of the scan class, in ms
execcount	integer	The number of times the scan class has executed
nextexec	datetime	The next time that the scan class should execute

sqlt_as

Column	Data Type	Notes
id	integer	Unique id of alert state
statename	string	Name of alert state
severity	integer /Severity enum	
low	double	Low setpoint
high	double	High setpoint
flags	integer / Alert Flags	Flags that dictate how the state acts
lotagpath	string	Path to Tag that provides low setpoint, if low driven flag is set
hitagpath	string	Path to Tag that Provides high setpoint, if high driven flag is set
timedeadband	double	Time deadband value
timedbunits	integer / TimeUnits enum	Time deadband units

sqlt_perm

Column	Data Type	Notes
tagid	integer	ID of Tag that the permission belongs to
rolename	string	Name of the role that this permission is applied to
accessrights	integer / AccessRights enum	Access rights for the given role on the given Tag

sqlt_drv

Column	Data Type	Notes
name	string	Name of the Tag drive
ipaddr	string	Address of browser server, blank or null if browsing isn't available
port	integer	Port of browse server

Column	Data Type	Notes
id	integer	Auto-incrementing unique id for the write operation
tagid	integer	ID of the Tag to write to
intvalue	integer	Value, if Tag has integer data type
floatvalue	double	Value, if Tag has float or real data type
stringvalue	string	Value, if Tag has string data type
datevalue	datetime	Value, if Tag has date data type
responsecode	integer / Write Responseenum	The state of the write request. When created, the response code should be set to 2 - Pending
responsemsg	string	Write error if operation failed
t_stamp	datetime	The time that the write request was created
objectid	integer	ID of the object with the error
objecttype	integer / Object Type enum	The type of object. Used with objectid to identify the item that caused the message
lifecycleid	integer/ Lifecycle enum	When the message was generated
msgtype	integer / Message Type enum	
errormsg	string	The primary message
stack	string	Additional error information
t_stamp	datetime	When the message was generated

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Tag Event Scripts



INDUCTIVE
UNIVERSITY

Tag Scripts

[Watch the video](#)

Tag Event Overview

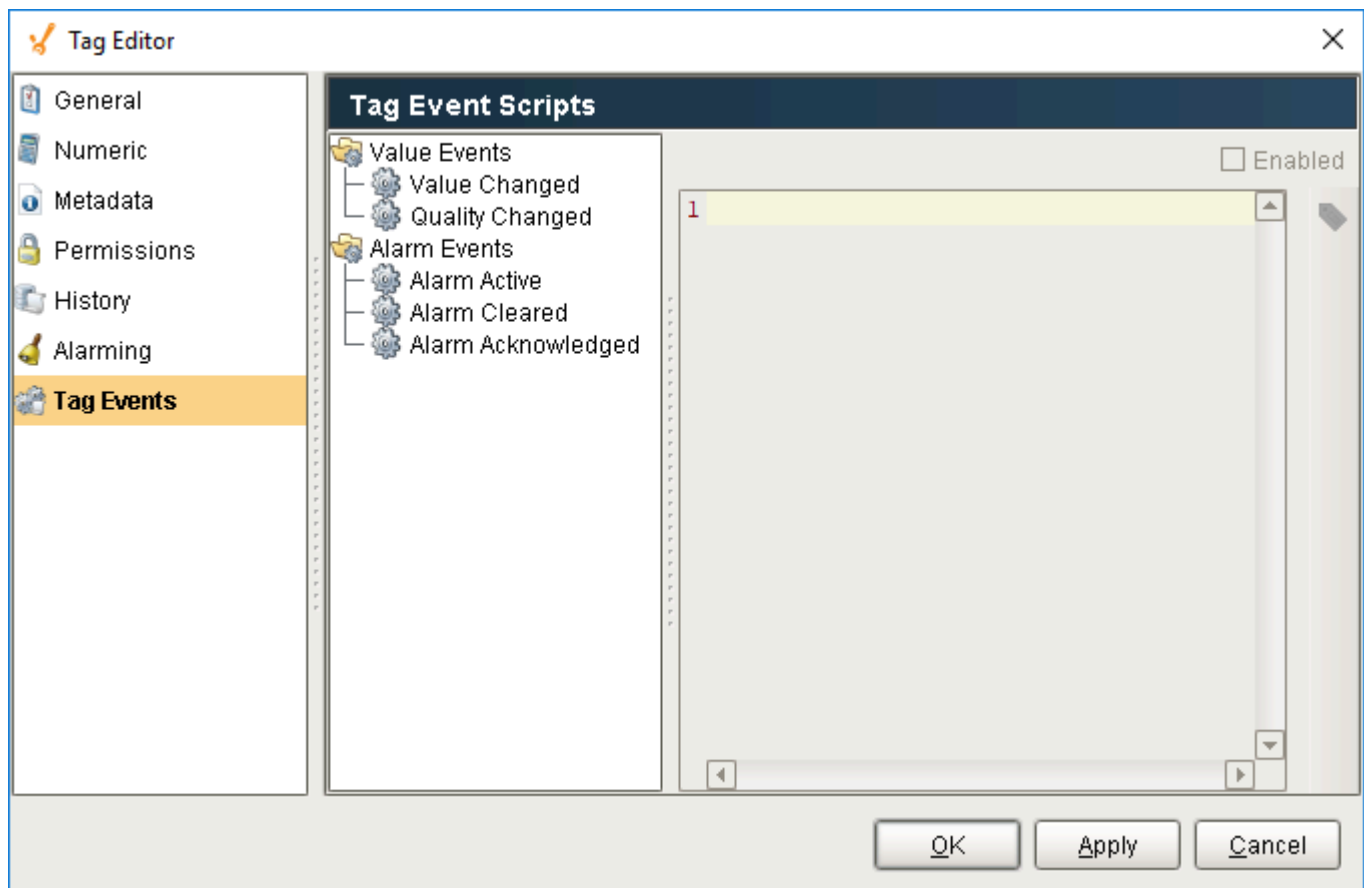
Scripts can be attached to Tags. When you edit a Tag, you can navigate to the Tag Events section to see a list of all of the available events. Those events are

- Value Changed
- Quality Changed
- Alarm Active
- Alarm Cleared
- Alarm Acknowledged

Each event is unique and can have specialized arguments depending on the event. Also, because Tags are stored in the Gateway, all these scripts fire in the **Gateway scope**.

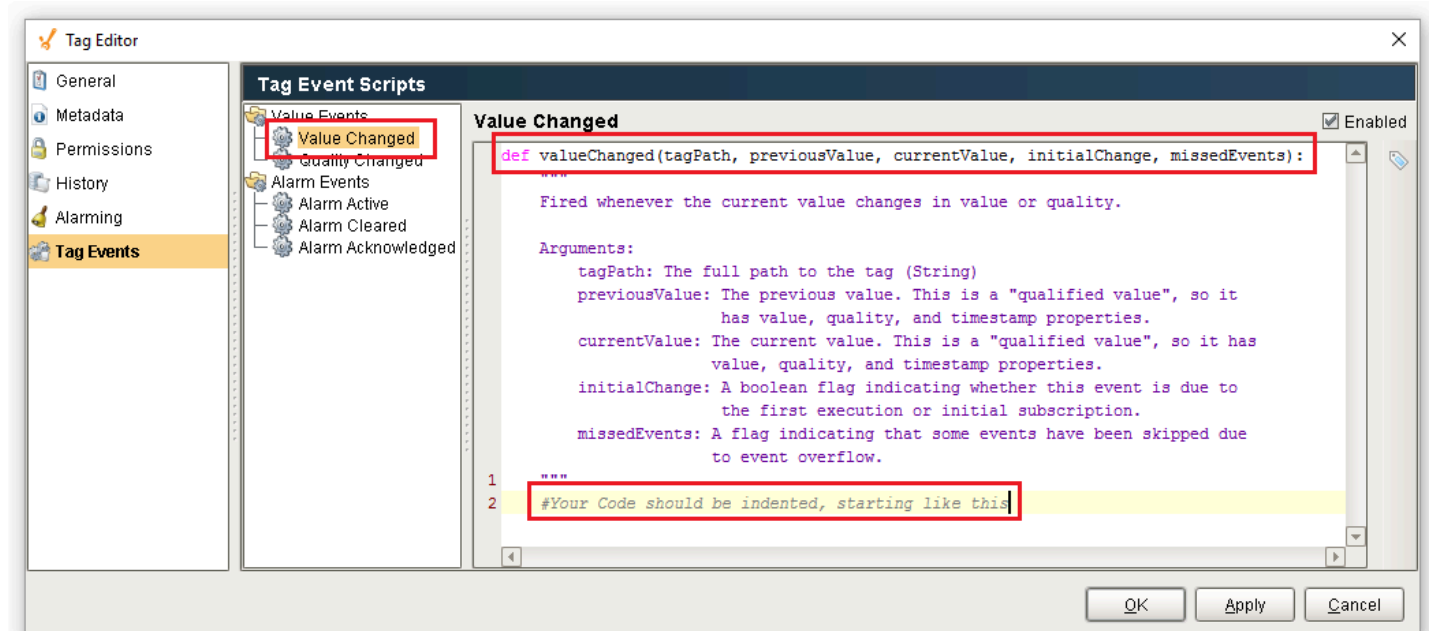
CAUTION

Because these scripts are Gateway scoped, certain things like print statements will not print to the Designer console, but will print instead to the wrapper log file in Ignition's installation directory. Keep that in mind when doing any testing.



Event Scripts are Functions

Once an event has been selected, note that the top of the text area is **defining a function**. As a result, all code that should execute when this event occurs should be indented at least once to be included in the definition.



Event Overview

Value Changed

The Value Changed event is fired whenever the value of this particular Tag changes. This event has a variety of arguments available for use in the script:

- **String tagPath** - The full path to the Tag. Object **previousValue** - The previous value. This is a qualified value, so it has value, quality, and timestamp properties.
- **Object currentValue** - The current value. This is a qualified value, so it has value, quality, and timestamp properties.
- **Boolean initialChange** - A boolean flag indicating whether this event is due to the initial subscription or the first execution after a Tag update.
- **Boolean missedEvents** - A flag indicating that some events have been skipped due to an event overflow.

! INFO

The `currentValue` and `previousValue` arguments are qualified values. This means that to get to the actual current value, it would look like `currentValue.value`.

Quality Changed

The Quality Changed event is fired whenever the quality of this particular Tag changes. Since Tags use a 'qualified value', which include a value, quality, and timestamp, this script will fire whenever any of those change. This event has a variety of arguments available for use in the script:

- **String tagPath** - The full path to the Tag.
- **Object previousValue** - The previous value. This is a qualified value, so it has value, quality, and timestamp properties.
- **Object currentValue** - The current value. This is a qualified value, so it has value, quality, and timestamp properties.
- **Boolean initialChange** - A boolean flag indicating whether this event is due to the initial subscription or the first execution after a Tag update.
- **Boolean missedEvents** - A flag indicating that some events have been skipped due to an event overflow.

Alarm Active

The Alarm Active event fires whenever an alarm becomes active on the Tag. This event has a variety of arguments available for use in the script:

- **String tagPath** - The full path to the Tag.
- **String alarmName** - The name of the alarm. This does not include the full alarm path.
- **Object alarmEvent** - The full alarm event object. The properties available to this object are:
 - id
 - source
 - name
 - priority
 - displayPath
 - displayPathOrSource (the display path if it is set, otherwise the source)
 - state (the current state, active/cleared + acked/unacked)
 - lastEventState (the last transition)
 - cleared (a boolean if the alarm is currently cleared)
 - acked (a boolean if the alarm is currently acknowledged)
 - shelved (a boolean if the alarm is currently acknowledged)
 - notes
- **String alarmPath** - The full alarm path.
- **Boolean missedEvents** - A flag indicating that some events have been skipped due to an event overflow.

Alarm Cleared

The Alarm Cleared event fires whenever an alarm becomes cleared on the Tag. This event has a variety of arguments available for use in the script:

- **String tagPath** - The full path to the Tag.
- **String alarmName** - The name of the alarm. This does not include the full alarm path.
- **Object alarmEvent** - The full alarm event object. See the Alarm Active alarmEvent object for the full list of available properties.
- **String alarmPath** - The full alarm path.
- **Boolean missedEvents** - A flag indicating that some events have been skipped due to an event overflow.

Alarm Acknowledged

The Alarm Acknowledged event fires whenever an alarm event becomes acknowledged on the Tag. This event has a variety of arguments available for use in the script:

- **String tagPath** - The full path to the Tag.
- **String alarmName** - The name of the alarm. This does not include the full alarm path.
- **Object alarmEvent** - The full alarm event object. See the Alarm Active alarmEvent object for the full list of available properties.
- **String alarmPath** - The full alarm path.
- **String ackedBy** - The full path to the user that acknowledged the alarm.
- **Boolean missedEvents** - A flag indicating that some events have been skipped due to an event overflow.

Examples

Printing all parameters

```
# This script will fetch all of the possible parameters in the tag changed script.  
# Note that this will not print out to the console, but it will print to the  
wrapper logs located on the gateway.
```

```
path = tagPath  
prev = previousValue  
cur = currentValue  
init = initialChange  
missed = missedEvents  
print path, prev.value, cur.quality, init, missed
```

Automatic Reset

```
# This code can be used on a Value Changed script, and will automatically reset  
the value of the tag to 0 after it goes above 3000.  
# This can be useful for counter tags.  
if currentValue.value > 3000:  
    system.tag.write("IntegerCounterTag", 0)
```

Copy to another Tag

```
# This code can be used on a Value Changed script, and will record the highest  
value of the current tag onto another memory tag.  
# This can be useful for recording the highest temperature.  
highestRecordedTemp = system.tag.read("HighestTempTag").value  
if currentValue.value > highestRecordedTemp:  
    system.tag.write("HighestTempTag", currentValue.value)
```

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Browsing and Creating OPC Tags

OPC Tags

There are two ways to create OPC tags.

- **From OPC Browser** - Using the OPC Browser to browse and find the tag and then drag it to the Tag Browser to create the tag. This method is the easiest and most common way to create tags.
- **From Tag Editor** - When the tag is not available for browsing from the OPC Browser, you create the tag manually, from the Tag Editor window.

Both of these methods require that you first have a [device connection](#) made.

Browsing for Tags



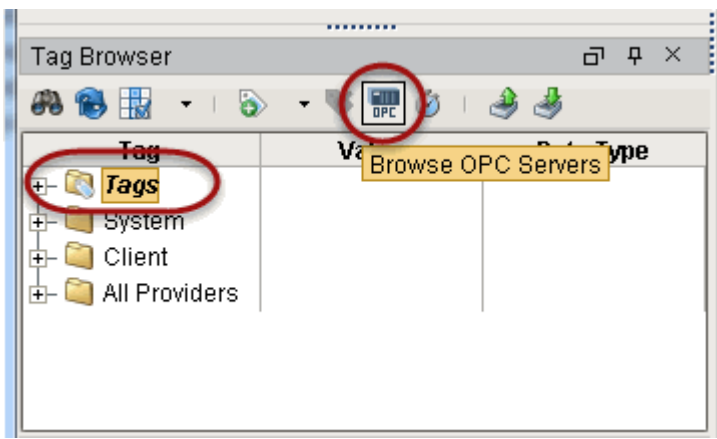
INDUCTIVE
UNIVERSITY

Browsing for OPC Tags

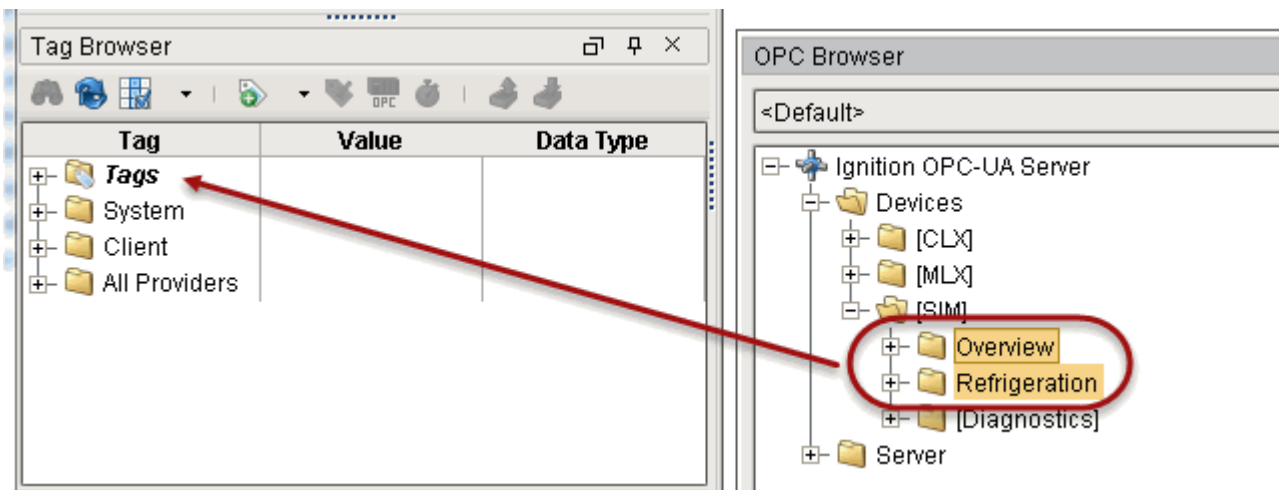
[Watch the video](#)

The easiest and most common way to create Tags in Ignition is by dragging the tags from the OPC Browser into the Tag Browser window in the Designer.

1. In the **Tag Browser**, select the **Tags** folder and click the **Browse OPC Servers** icon. The OPC Browser is displayed and you can browse all of your OPC connections. By default, you've got a connection to the internal **Ignition OPC-UA Server**, which has any devices that may be connected. Browse the devices and find some tags that you're interested in.



- From the **OPC Browser**, highlight the tags (for example, as in the picture below, both folders inside the **SIM** folder) and drag them to the **Tags** folder in the **Tag Browser**. You can drag individual tags or folders. When you drag folders, Ignition keeps the same hierarchy as the PLC. As soon as you drag the tags into Tag Browser, you can see their values being updated in the Tag Browser.



You now have some Tags. You can see their values come in and start updating automatically. By default, they update at a rate of 1 per second.

To Create OPC Tags Manually



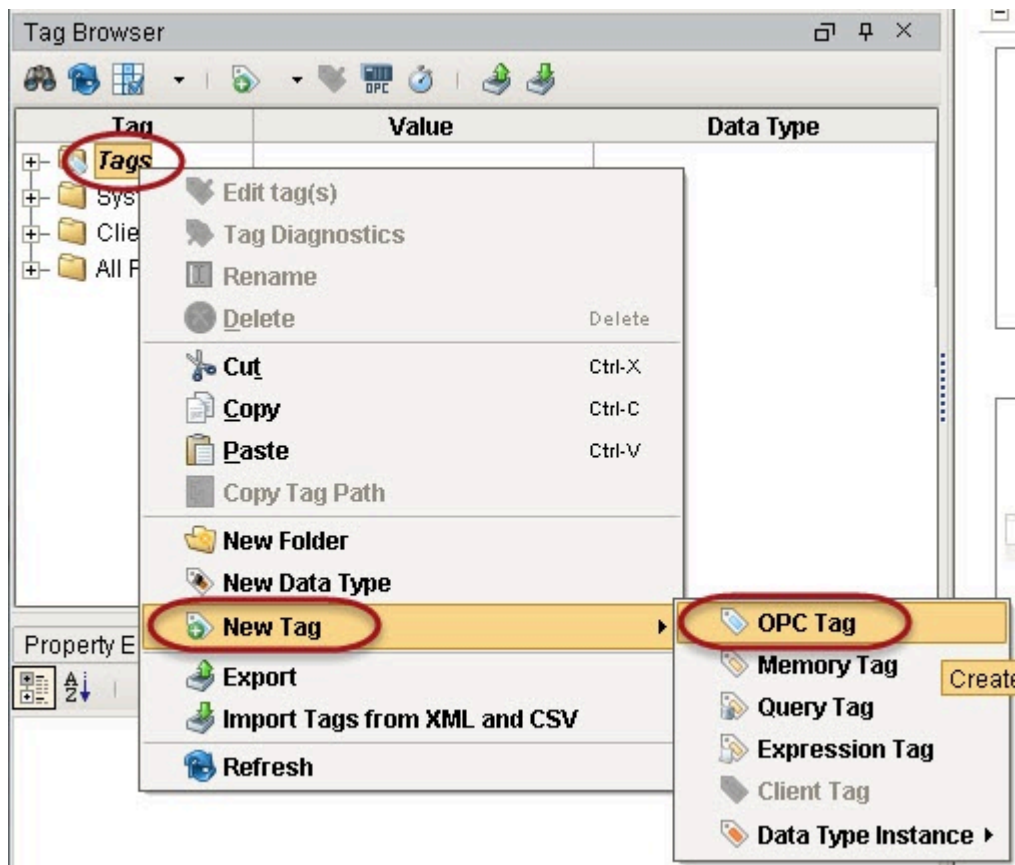
INDUCTIVE
UNIVERSITY

Creating OPC Tags Manually

[Watch the video](#)

You can create OPC Tags manually when you can't find them using the OPC Browser, or the device does not support browsing.

- From **Tag Browser**, right-click on **Tags** or a folder of your choice and select **New Tag > OPC Tag**.



2. In the **Tag Editor** window, as an example, you can set the following values:

- Name: **Temperature**
- Data Type: **Int4**
- OPC Server: choose **Ignition OPC-UA Server** from the dropdown
- OPC Item Path: **[Modbus]HR1**, the **Modbus** device name goes in the square brackets then you give the address to PLC which in this case is the **HR** designator plus **1** as the Modbus address. The Modbus Specific Addressing section, explains how your can construct these addresses.

The screenshot shows the 'Tag Editor' window with the 'General' tab selected. The 'General Properties' section includes fields for 'Name' (Temperature), 'Data Type' (Int4), 'Enabled' (Yes), and 'Access Rights' (Read/Write). The 'OPC Properties' section includes 'OPC Server' (Ignition OPC-UA Server) and 'OPC Item Path' ([Modbus]HR1). The 'Scan Class' section shows 'Default'. Red boxes highlight the 'Name' and 'OPC Properties' sections.

3. Click **OK**.

Now you can see the **Temperature** tag in the **Tag Browser**.

Addressing Bits



INDUCTIVE
UNIVERSITY

Addressing Bits

[Watch the video](#)

In order to address individual bits in Ignition, you must create a separate OPC tag pointing directly to the specific bit in the PLC.

When the integer values that come from the OPC tags are a series of binary bits, it is then possible to address each bit. For example, an integer value can have a 16-bit binary representation as shown here:

Integer	Bit level representation	How it works
4096	0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0	$2^{12} = 4096$
1025	1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0	$2^0 + 2^{20} = 1025$

To Address an Individual Bit

Suppose a MicroLogix PLC is connected to a Gateway.

- From the **Tag Editor** window, create an OPC tag to have:
 - Data Type: **Integer** with a value of 1025
 - OPC Item Path: `[MLX]B3:0`
- Then create a new OPC tag with a Boolean value to the first bit of this tag as follows:
 - Data Type: **Boolean** with a value of 1 or True
 - OPC Item Path: `[MLX]B3:0/0` (for Micrologic, you can specify the bit as: 0/0 or 0.0. That is, with a slash/ or a period.) `[MLX]B3:0/0` has a value of 1 or a Boolean value of True.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Creating Memory, Expression, and Query Tags

Memory Tags



INDUCTIVE
UNIVERSITY

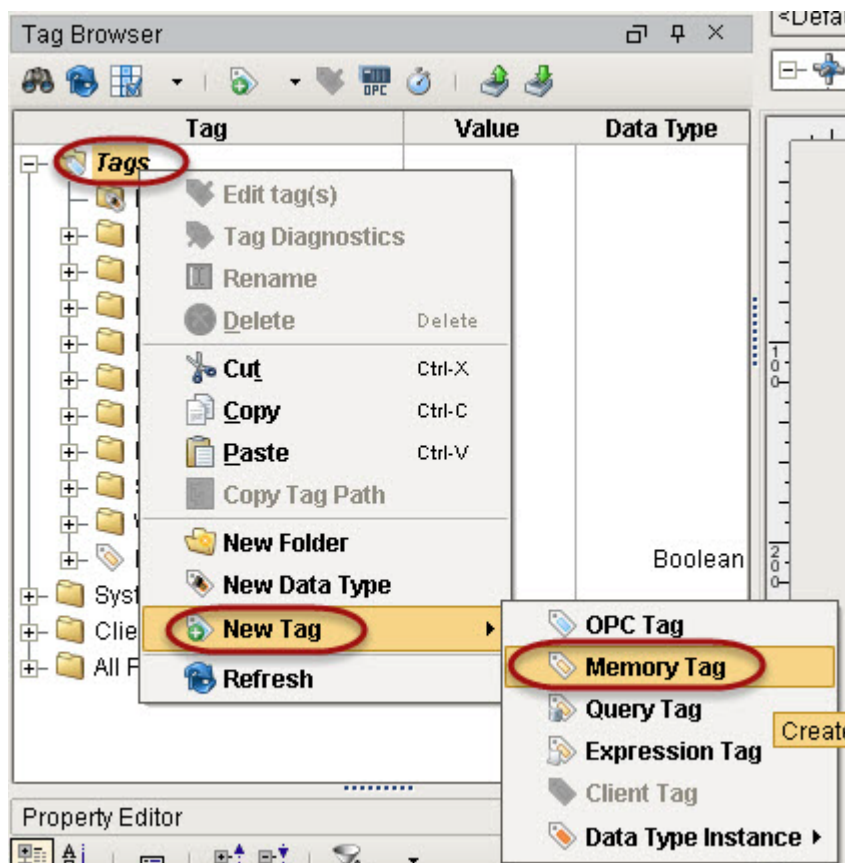
Creating Memory Tags

[Watch the video](#)

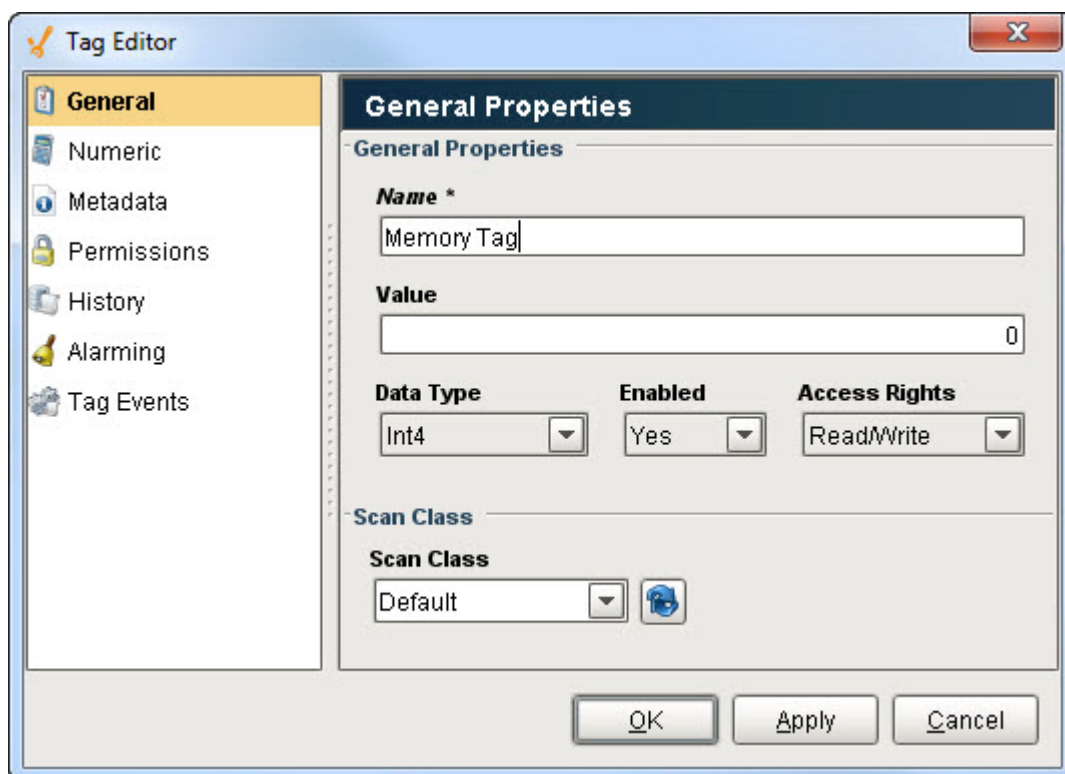
Memory tags are simply values. You can use them as setpoints (that are not stored in the PLC) or just for storing extra information. The value is specified during configuration and is stored globally in the gateway when written.

To Create Memory Tags

1. From **Tag Browser**, right-click on **Tags** or a folder of your choice and select **New Tag > Memory Tag**.



2. In the **Tag Editor**, type a **Name** (for example, Memory Tag), initial **Value**, and **Data Type**.



3. Click **OK**.

You now have a memory tag that you can read from or write to. This is just like having a global variable for your project.

Creating Expression Tags



INDUCTIVE
UNIVERSITY

Creating Expression Tags

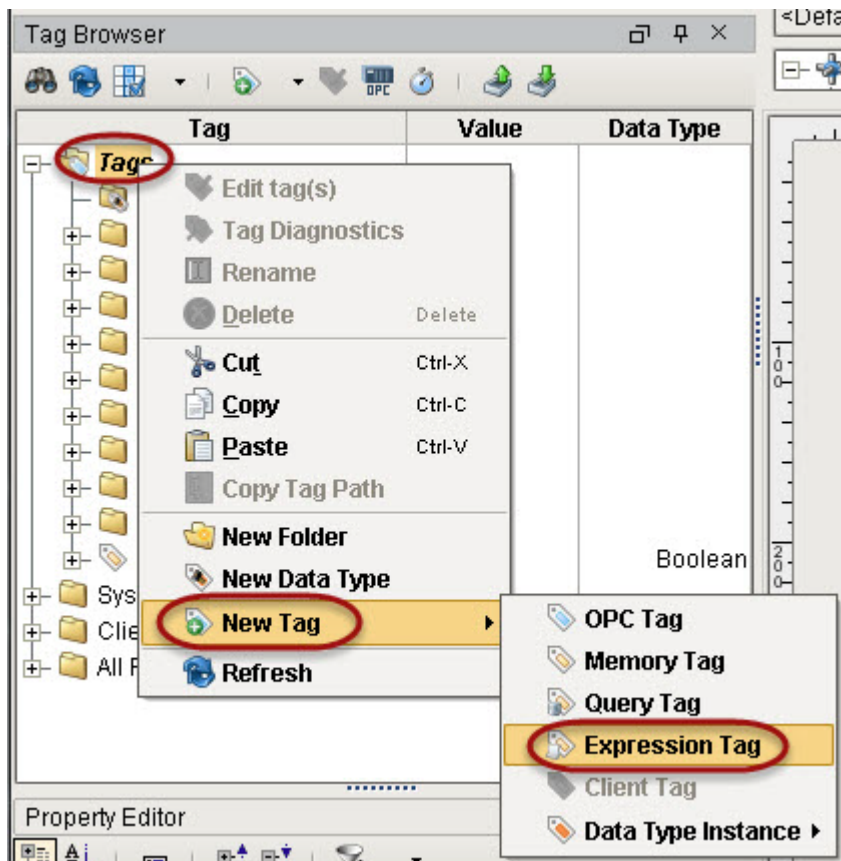
[Watch the video](#)

Expression Tags are driven by an expression. The expression syntax is the same as for property bindings and allows mathematical operations, references to other tags, logic operations, and more. Expression Tags cannot be written to.

With expression tags you can do a calculation based on certain values, other tags, or any one of the built in functions. In this way, you can do the calculation one time and view it on any window in your project.

To Create an Expression Tag

1. In the **Tag Browser**, right-click on **Tags** or a folder of your choice and select **New Tag > Expression Tag**.

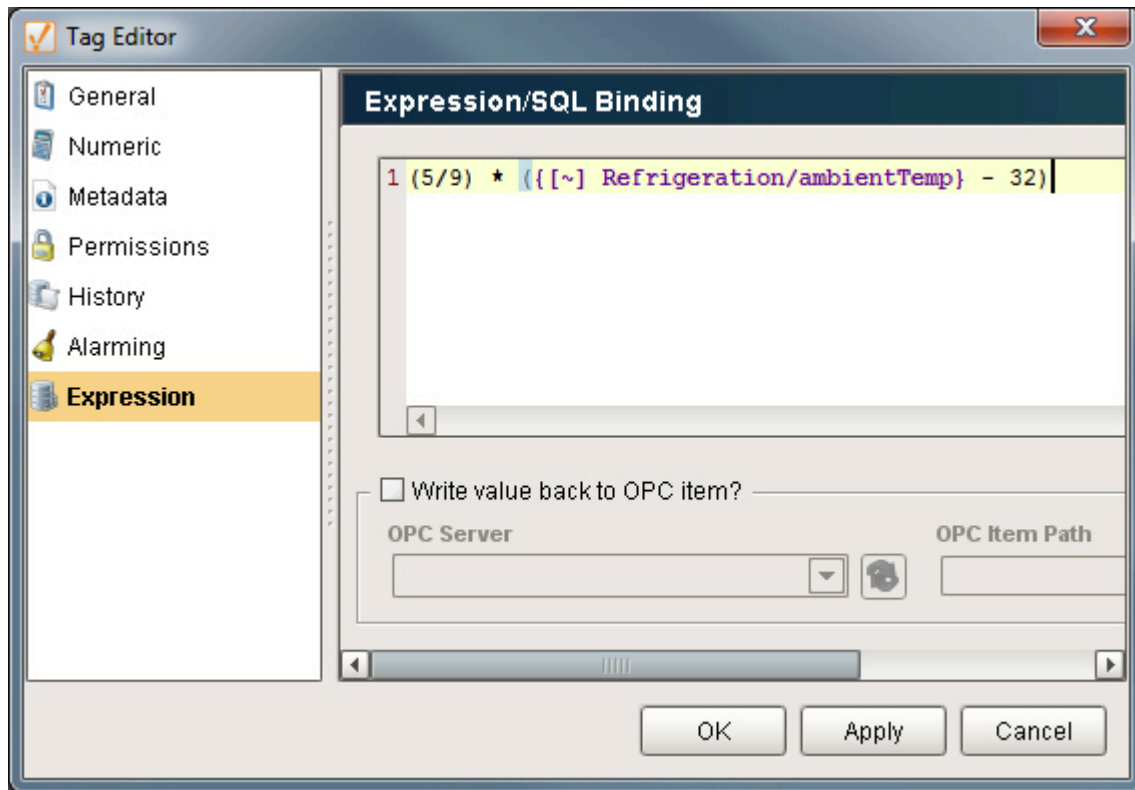


2. In the **Tag Editor**, type **FtoC Expression** for **Name** and choose **Float4** for **Data Type**.
3. Click on the **Expression** tab on the left side of **Tag Editor**.

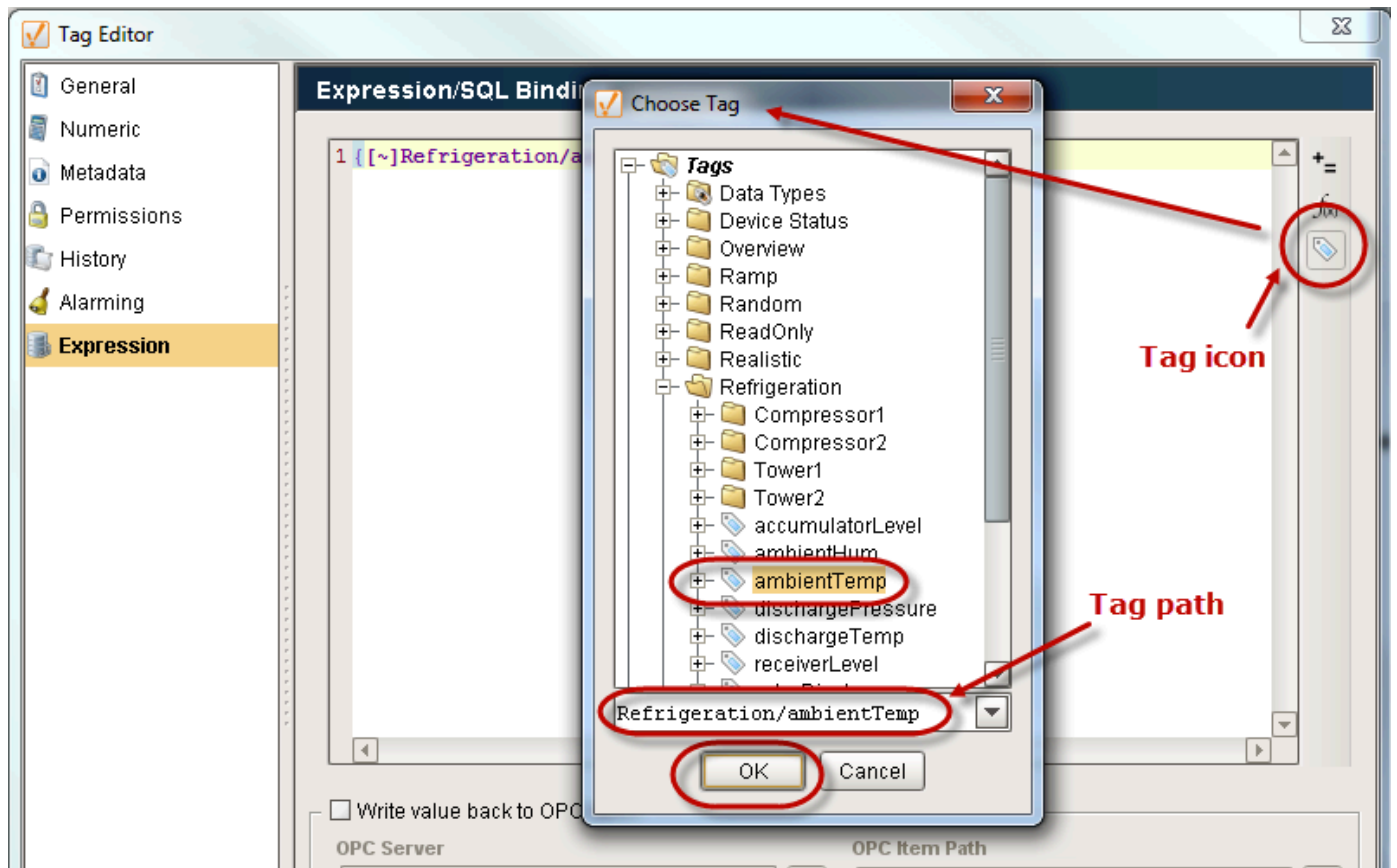
4. Type in any expression using Ignition's [Expression Language](#). See Expression Functions in the Appendix for information on all expression operations and functions.

Copy in the following expression:

```
(5/9) * ({[~]Refrigeration/ambientTemp} - 32)
```



If you want to use a different tag, just replace the `{[~]Refrigeration/ambientTemp}` tag path with any Tag of your choice. Note that instead of typing the referenced tag path (in this example `Refrigeration/ambientTemp`), you can use the Tag icon on the right side of the **Tag Editor > Expression** window to select the tag path.



5. Click **OK**. You will see the value from your calculation on the Tag. Note that the Expression tags are evaluated at the rate the Scan Class is set to.



With an expression tag, you can also choose to write the value back to another OPC item. In the **Tag Editor > Expressions**, simply click the check box next to **Write value back to OPC item?** and select the tag you want to write to. Every time the value is calculated, the value gets written to the selected tag.

Creating Query Tags



INDUCTIVE
UNIVERSITY

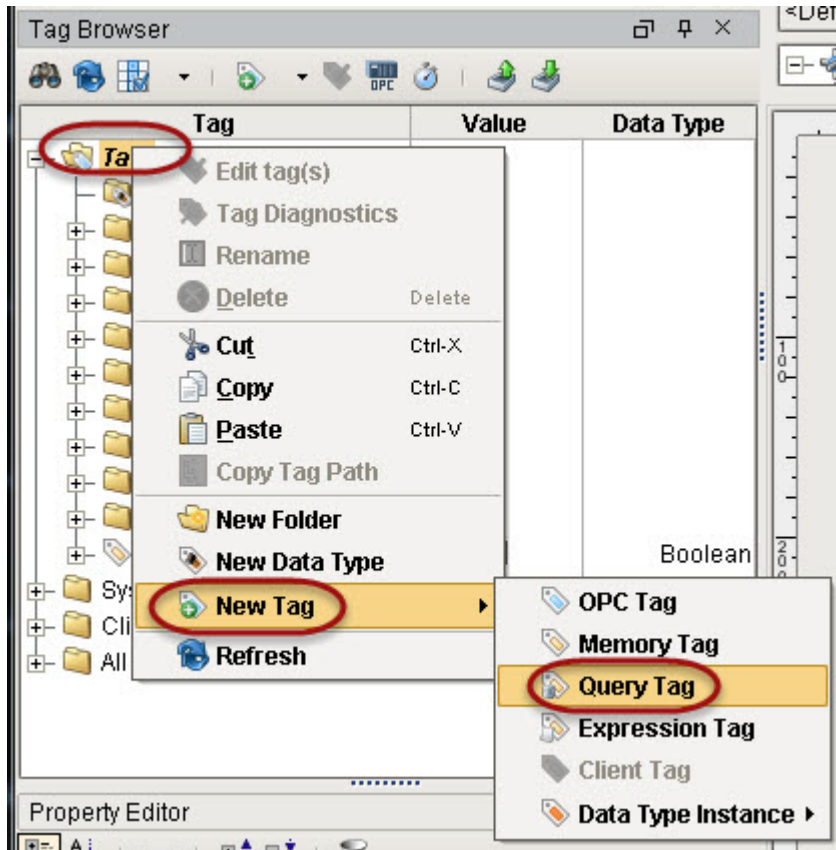
Creating Query Tags

[Watch the video](#)

Query Tags execute a SQL Query, whose result provides the value for the Tag. Like SQL binding in Vision, SQL Query Tags can reference other tags to build dynamic queries. Query Tags cannot be written to. The value of a SQL Query tag comes from the database through a SQL query.

To Create a SQL Query Tag

1. In **Tag Browser**, right-click on **Tags** or a folder of your choice and select **New Tag > Query Tag**.



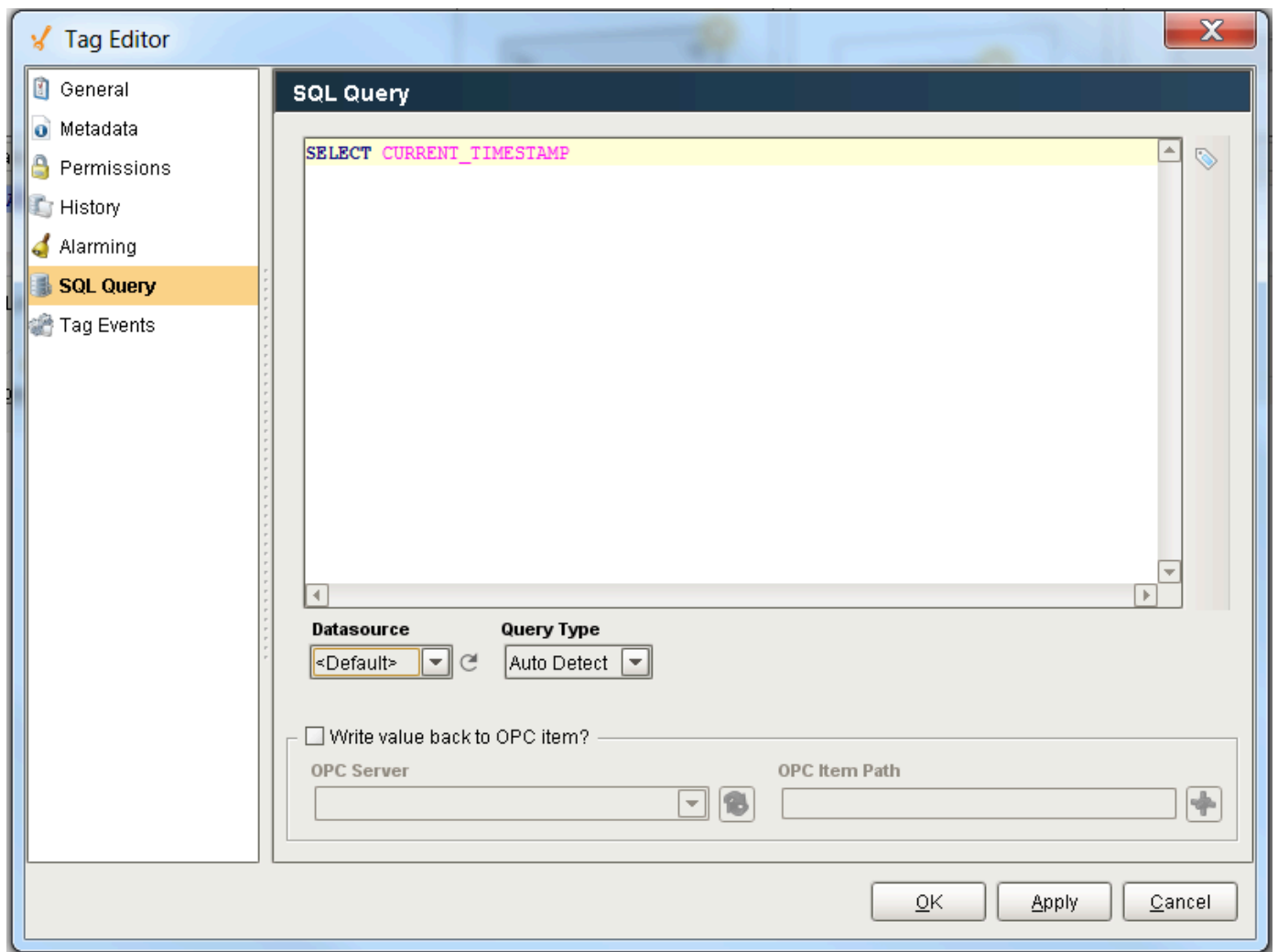
2. In the **Tag Editor** window, type **Current Time** for **Name** and choose **DateTime** for **Data Type**.
3. Click on **SQL Query** tab on the left side of **Tag Editor**.
4. Type in any SQL query you want.

i NOTE

The query must return a scalar result because the data type is not a dataset.

5. In the drop-down list, select the database to run the query through, for example **MySQL**.
6. Copy in the following query:

```
SELECT CURRENT_TIMESTAMP
```



! QUERY TYPE

In certain circumstances you may want to update the database instead of querying it for data. The query tag will accept an update query and update the target data source. The value attribute of the tag becomes the number of rows affected by the update query.

The data source refers to the default data source of the tag provider as opposed to the default data source of the project.

In the Gateway **Configure** Section, go to **Tags > Realtime** and edit the appropriate tag provider. Set the default database connection that this tag provider will use for its query tags.

7. Click **OK**.

You will see the value from your SQL query on the **Current Time** Tag.

SQLTags Browser		
		
Tags		
+ Data Types		
+ Device Status		
+ Overview		
+ Ramp		
+ Random		
+ ReadOnly		
+ Realistic		
+ Refrigeration		
+ Sine		
+ Writeable		
+  Current Time	2014-04-08 11:24:22 AM	DateTime
+ System		
+ Client		
+ All Providers		

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Creating User Defined Types - UDTs



INDUCTIVE
UNIVERSITY

**Understanding Complex Tags
(UDTs)**

[Watch the video](#)

What is a UDT?

UDTs (User Defined Types) are extremely important in Ignition. UDTs, also referred to as Complex Tags, offer the ability to leverage object-oriented data design principles in Ignition. Using UDTs, you can dramatically reduce the amount of work necessary to create robust systems by essentially creating parameterized "data templates".

By defining UDTs and using these essentially "data templates", you can generate Tag instances to rapidly build complex screens. A change to the type definition is then inherited by all instances, drastically saving time when making routine changes.

The UDT data types are fully supported by [Vision Templates](#), which means you can configure templates for your custom data types and take advantage of drag-and-drop binding to rapidly build complex screens.

Primary UDT Features

Object Oriented

Use small or large groups of Tags to create a single object. Create objects that match your real world devices or the existing structures in your PLCs.

Central Definition

Once you define your data type, you can then create instances of it. If at a later time you want to change some aspect of the type, you can simply edit the type definition, and all instances of it are automatically updated.

Parameterized Settings

Define custom parameters on your data type, and then reference them inside some or all of your member Tags. When it comes time to create instances, you can simply modify their parameter values in order to change where the underlying data comes from.

Extendable

Data types can inherit from other data types in order to add additional members or override settings. Instances can also override settings, allowing for flexibility when dealing with irregularities and corner cases.

UDT Terminology

Many terms are frequently used when discussing complex Tags.

UDT

User Defined Type - the definition of the data type is its structure, Tags, attributes, and settings.

Instances

Instances are running copies of a data type with specific data for all members. Instances are linked to their parent types and are reloaded when their parent type changes. Besides specifically overridden settings, all settings are inherited from the type definition.

Parameters

Parameters are custom properties on data types that you can use inside the type or instance definition to create parameterized data templates. For example, if a data type consists of 3 OPC tags that only differ by a number in the path, you can use a parameter for the "base address", allowing instances to be created with only 1 setting.

Members

Members are the Tags inside of a data type or instance.

Defining UDTs

You can save time and quickly generate data types from existing structures. This is particularly useful, for example, when data types are already defined in the PLC. Creating a new data type is very similar to creating standard Tags. You can open the Tag Browser and drag multiple Tags or folders into the Data Types folder, or you can create them manually by right-clicking and selecting New Tag.

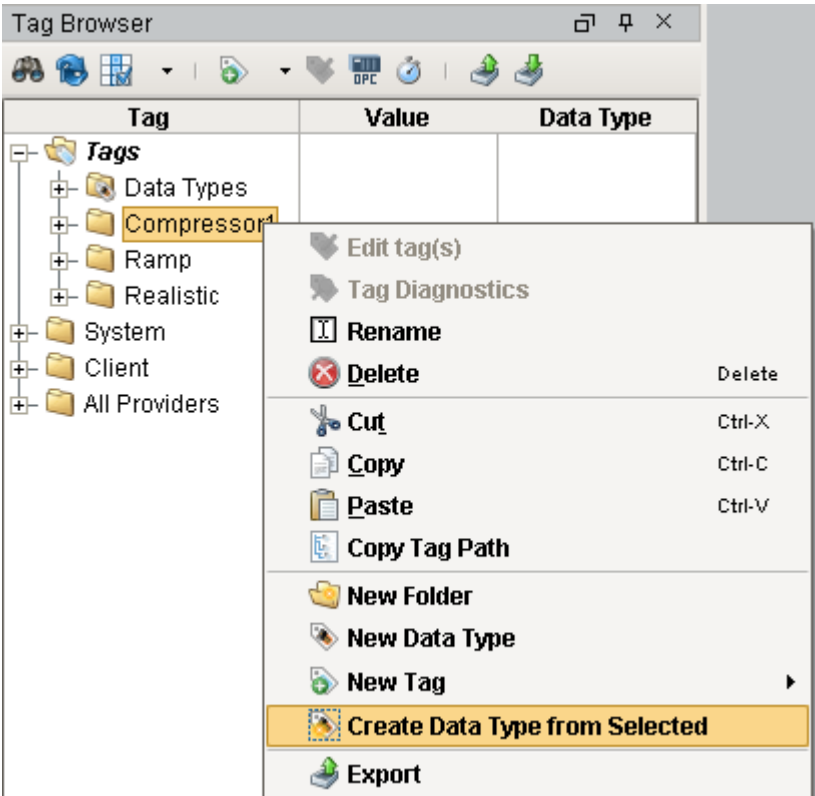
Converting from Existing Tags

To create a data type from existing Tags, simply select the Tags or folders you wish to include, right-click and select **Create Data Type from Selected**. The Tag Editor window is displayed with the selected tags pre-populated as members. From here you can modify the Tags, add parameters, and so on. The original Tags will not be affected.



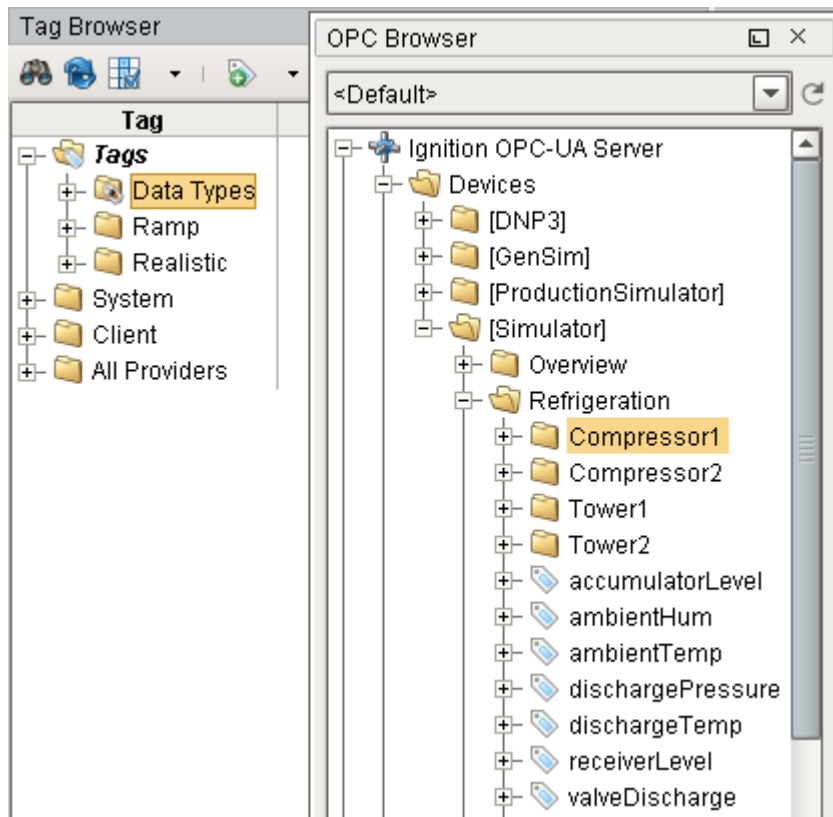
TIP

If you select a single folder as the root to create the type from, its sub-members will be added, and its name will be the basis for the type (that is, the folder itself won't be included in the structure).



Creating from OPC

If you have data types defined in your PLC (or OPC server), you can short-cut the conversion step outlined above by simply dragging the Tags (or the folder for the type) from the **OPC Browser** directly onto the **Data Types** folder, and selecting **Create Type** from the subsequent dialog.

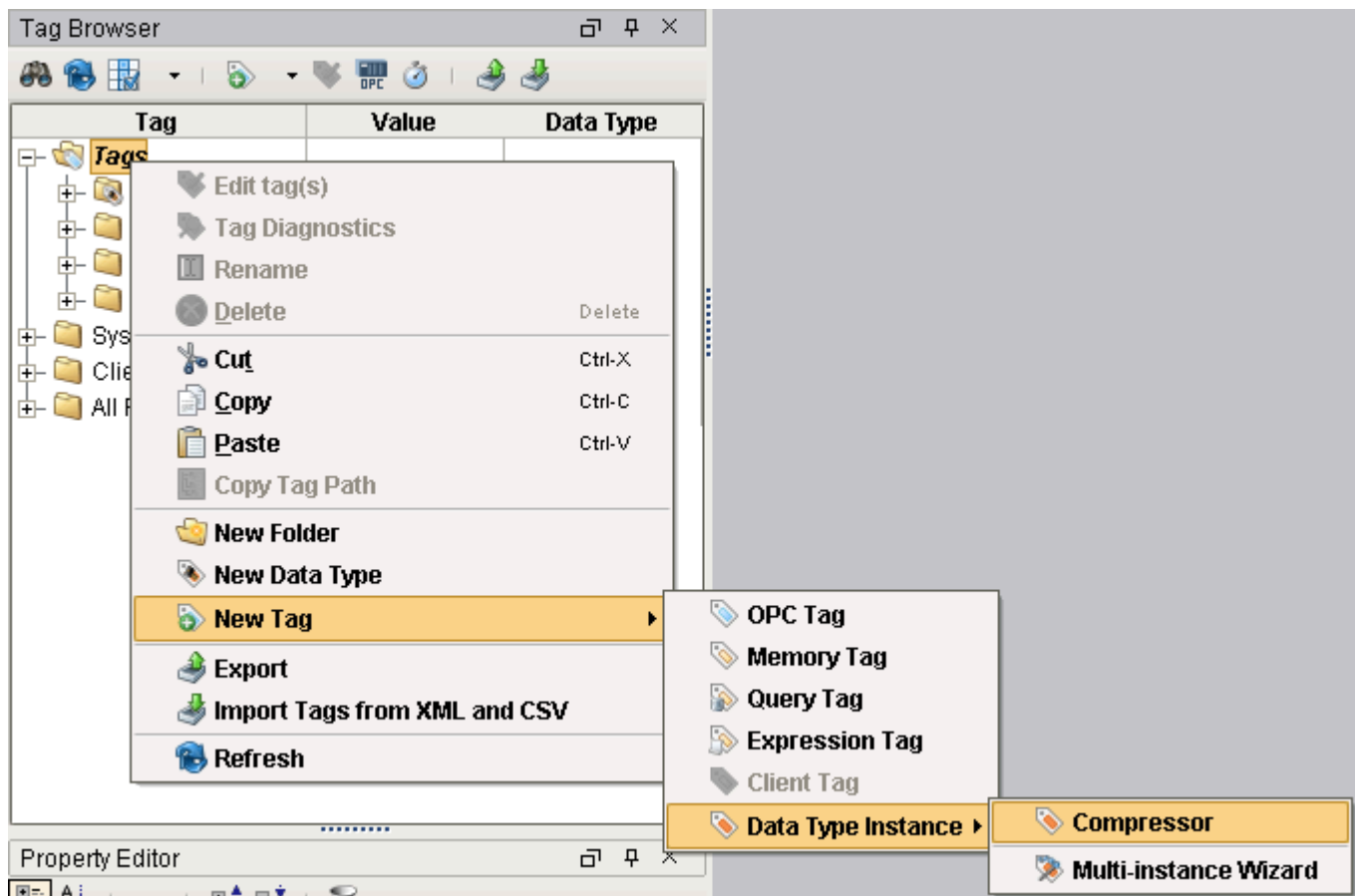


Creating Instances of UDTs

Creating instances of complex types is virtually identical to creating other types of Tags using the New Tag menu. Unlike standard Tags, it is likely that you'll have to modify attribute values or override certain member properties in order to make the instance unique. The process for doing this is the same to that used when creating data types. Once created, instances run very much like standard Tags. If the parent data type is updated, the instance will automatically receive the updates and refresh.

Creating Instances Manually

You create individual instances of a UDT in exactly the same way as a basic Tag. Right-click and select **New Tag**, then the type of UDT you want to create.



Using the Multi-Instance Wizard

The Multi-Instance Wizard provides a powerful but simple mechanism for rapidly generating many instances of a data type, by specifying patterns for parameters. To get started, right-click a Tag in the **Tag Browser**, then select **New Tag > Data Type Instance > Multi-instance Wizard** from the menu. The **Instance Creation Wizard** window is displayed. Once you select a data type, you will see its parameters come into the table, where you can specify values for them. The size of the parameter patterns dictate how many Tags get created, or if only single values are specified, you can choose to create multiple copies of the same configuration.

Instance Creation Wizard

Step 1 - Select Data Type to Create

Compressor

Step 2 - Configure Parameters

Base Tag Name:

Compressor

Tag Name Pattern:

1-10

Parameter Patterns

Parameter	Type	Pattern	Size
CompressorNumber	String	1-10	10

Tags to create: 10

The pattern is used to generate values, and can be:
Range: *x-y[/step]*, for numerical values. Example: *0-10*, *0-100/10*
List: *a,b,c* Example: *West,East,North,South*
Repeat: *value*count* Example: *1*10*
List patterns can also contain range and repeat patterns as values. For example, *0-10,20-30,40*10*

Number padding can also be used. For example, *000-005* will result in "000, 001, ..."

Preview

OK

Cancel

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Creating UDT Definition



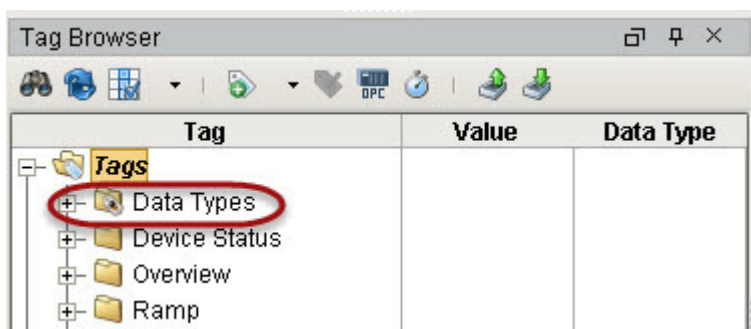
INDUCTIVE
UNIVERSITY

Creating UDT Definition

[Watch the video](#)

So far, we have only dealt with simple Tags. In Ignition, you can also create Complex Tags or User Defined Types (UDTs). These types can model UDTs in certain PLCs (such as a ControlLogix) or can be completely new.

You create the UDTs in a special folder in the **Tags Browser** called **Data Types**.



UDTs (User Defined Types) can be created in the following different ways:

- Browsing OPC servers via OPC Browser
- Creating UDTs from existing Tags
- Creating UDTs manually

If your PLC supports UDTs, the easiest way to create a user defined type is from OPC.

⚠ THE DATA TYPE FOLDER

When creating a new data type, the new definition is **always** placed in the Data Types folder, regardless of which folder was selected before creating the new definition.

Additionally, the Data Types Folder in the Tag Browser can only ever contain UDT definitions, and UDT definitions can only ever exist in the Data Types folder.

To Create the UDT from OPC Browser

In the example, we will use a ControlLogix which supports UDTs. We will use the second method from above, that is, creating the UDT from OPC. In our ControlLogix PLC, let's say we already have a motor UDT setup, and now we will create the UDT in Ignition.

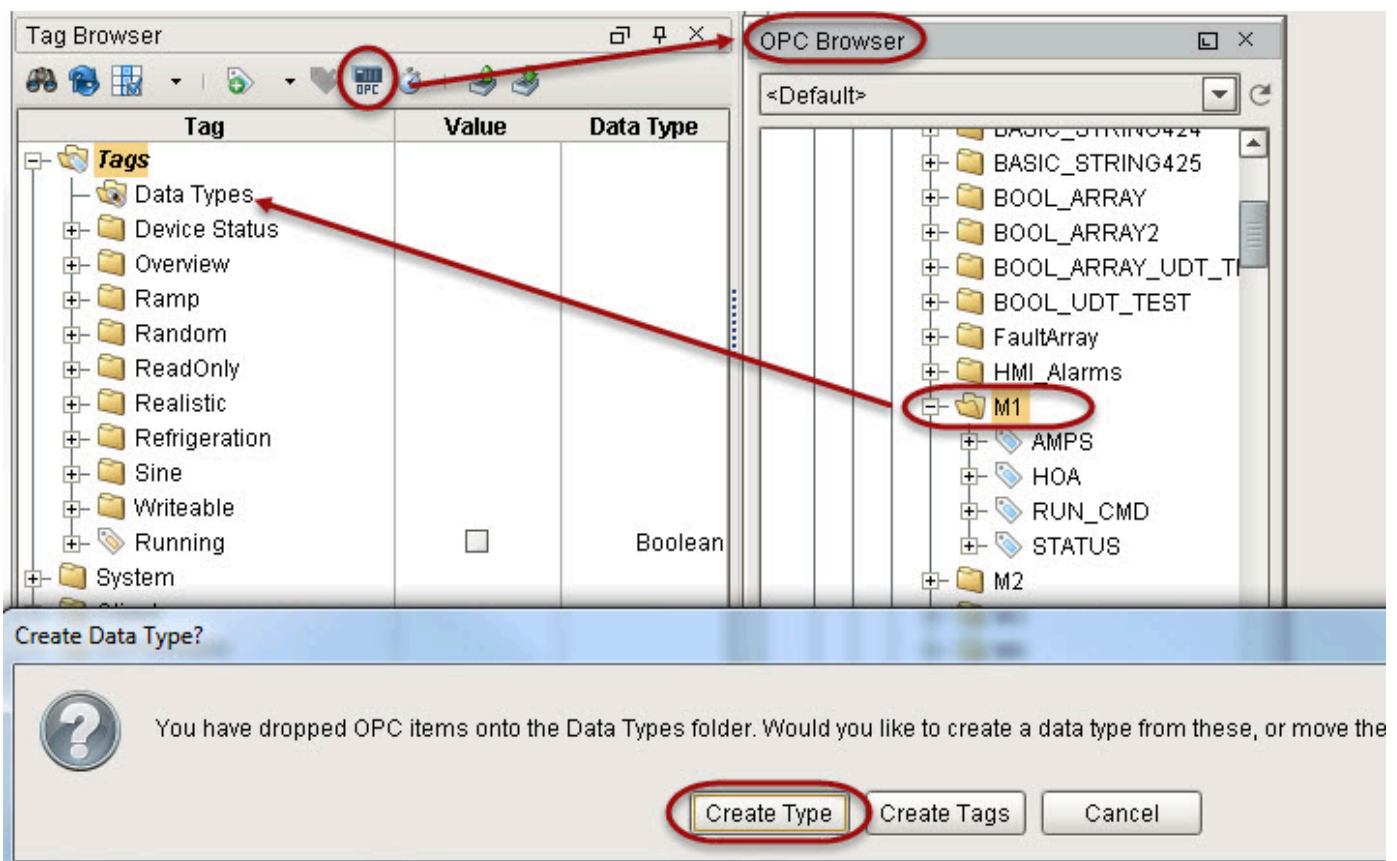
1. From **Tags Browser**, click the **OPC Browse** icon.
2. Under the **CLX** folder, go to the **Global** folder and find the **M1** folder.

M1 represents a motor which has four Tags:

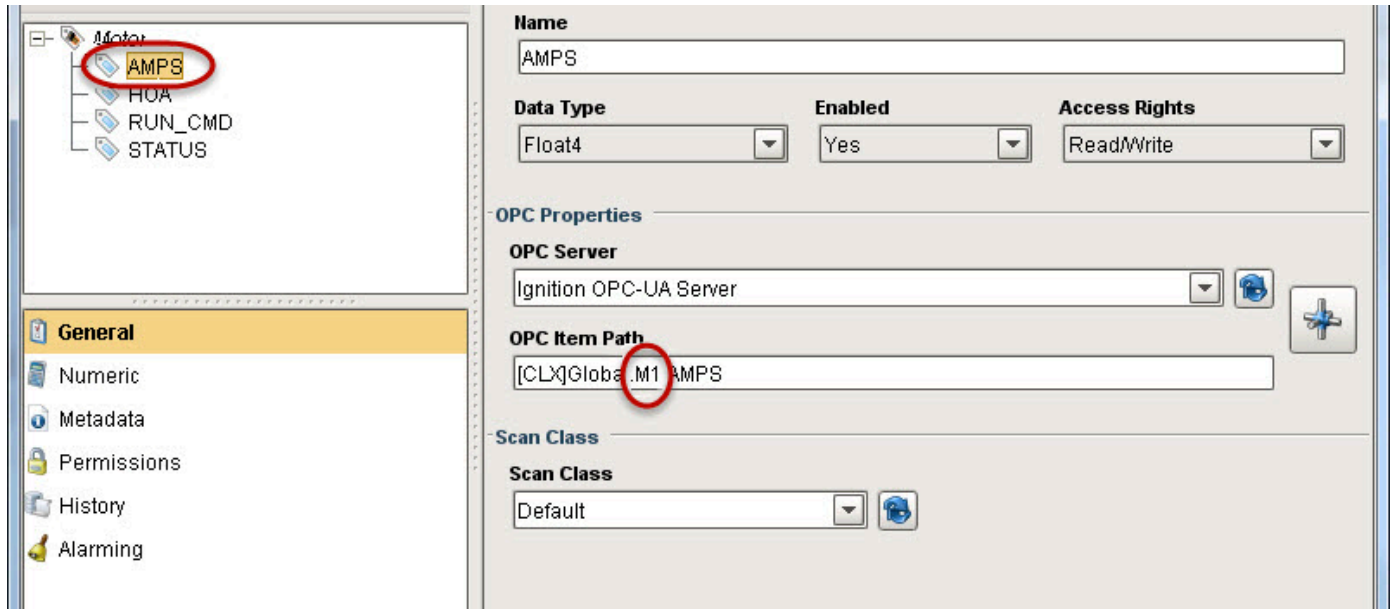
- AMPS
- HOA
- RUN_CMD
- STATUS

3. Drag the **M1** folder to the **Data Types** folder in **Tags Browser**.

A window prompts you asking if you want to create a type or Tags. Click on **Create Type**.



4. The **Tag Editor** window will open. Notice all four Tags; AMPS, HOA, RUN_CMD, and STATUS are automatically part of the UDT.
5. In the **Tag Editor**, change the name from **M1** to **Motor** to make the type name more of a generic name.
6. Right now, each Tag is pointing to a specific address in the PLC, select the **AMPS** Tag to see that it is pointing to **M1** in the PLC.

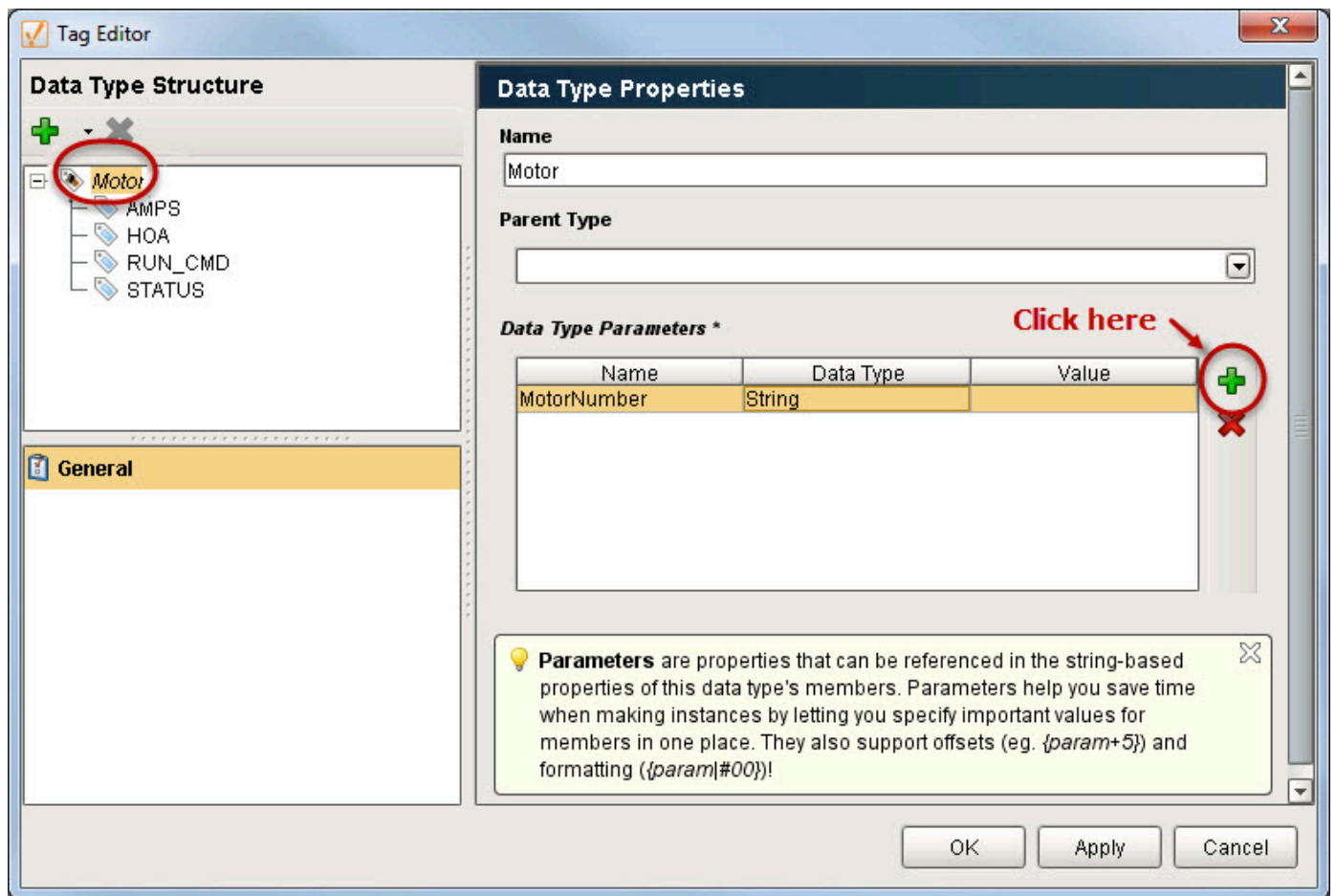


Because we are creating a UDT, we don't want to point to one specific set of Tags. We want each instance of the UDT to reference a different set of Tags. To do that, we need to add a parameter to the UDT. Parameters are custom properties on data types. You can use the parameters inside the type or instance definition to create parameterized data templates.

For example, if a data type consists of three OPC Tags that only differ by a number in the path, you can use a parameter for the “base address,” allowing instances to be created with only one setting.

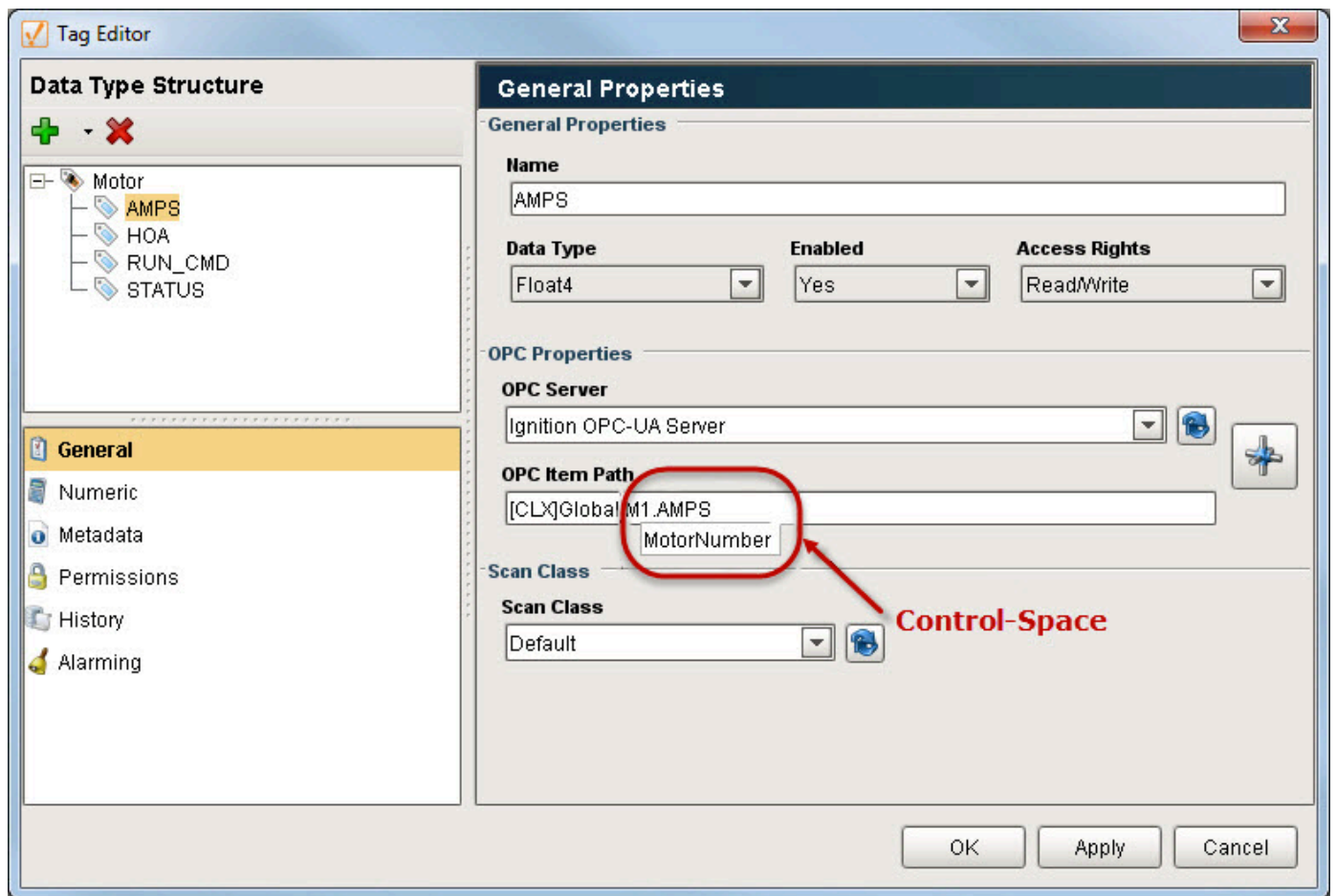
7. In the **Tag Editor**, select the Motor data type and add a new parameter by clicking the **green +** button and setting the following:
 - Name: **MotorNumber**
 - Data Type: **String**

Click **OK**.



8. Now for each Tag, we can substitute the motor number with the **MotorNumber** parameter.

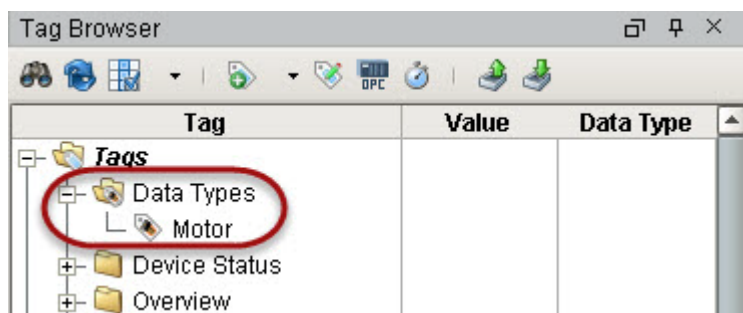
Open the **Tag Editor** again, select the **AMPS** Tag. In the **OPC Item Path**, put your cursor at the end of **M1** and press **Ctrl-Space** (or the Down arrow key) to bring up the list of parameters, click **MotorNumber** to add a reference `{MotorNumber}` into the OPC Item path. Don't forget to delete the '1' from **M1** before moving to the next Tag.



9. Repeat Step 8 for the **HOA**, **RUN_CMD**, and **STATUS** Tags.

The **OPC Item Path** will then show the **MotorNumber** parameter for each of the Tags.
Click **OK** to save the UDT.

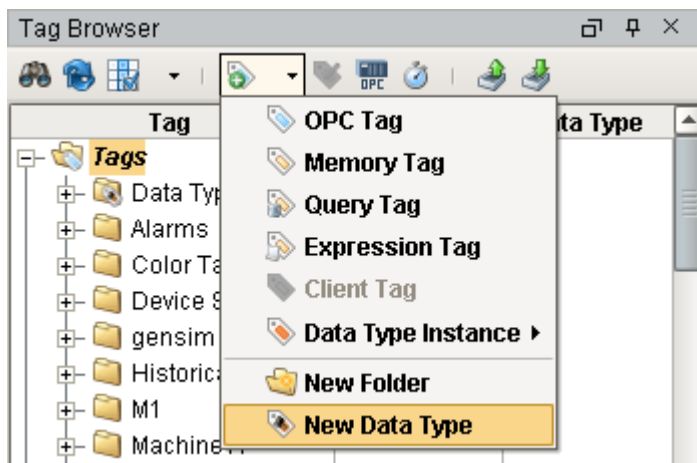
10. You will then see the **Motor** UDT in the **Data Types** folder.



To Create a New Data Type Manually

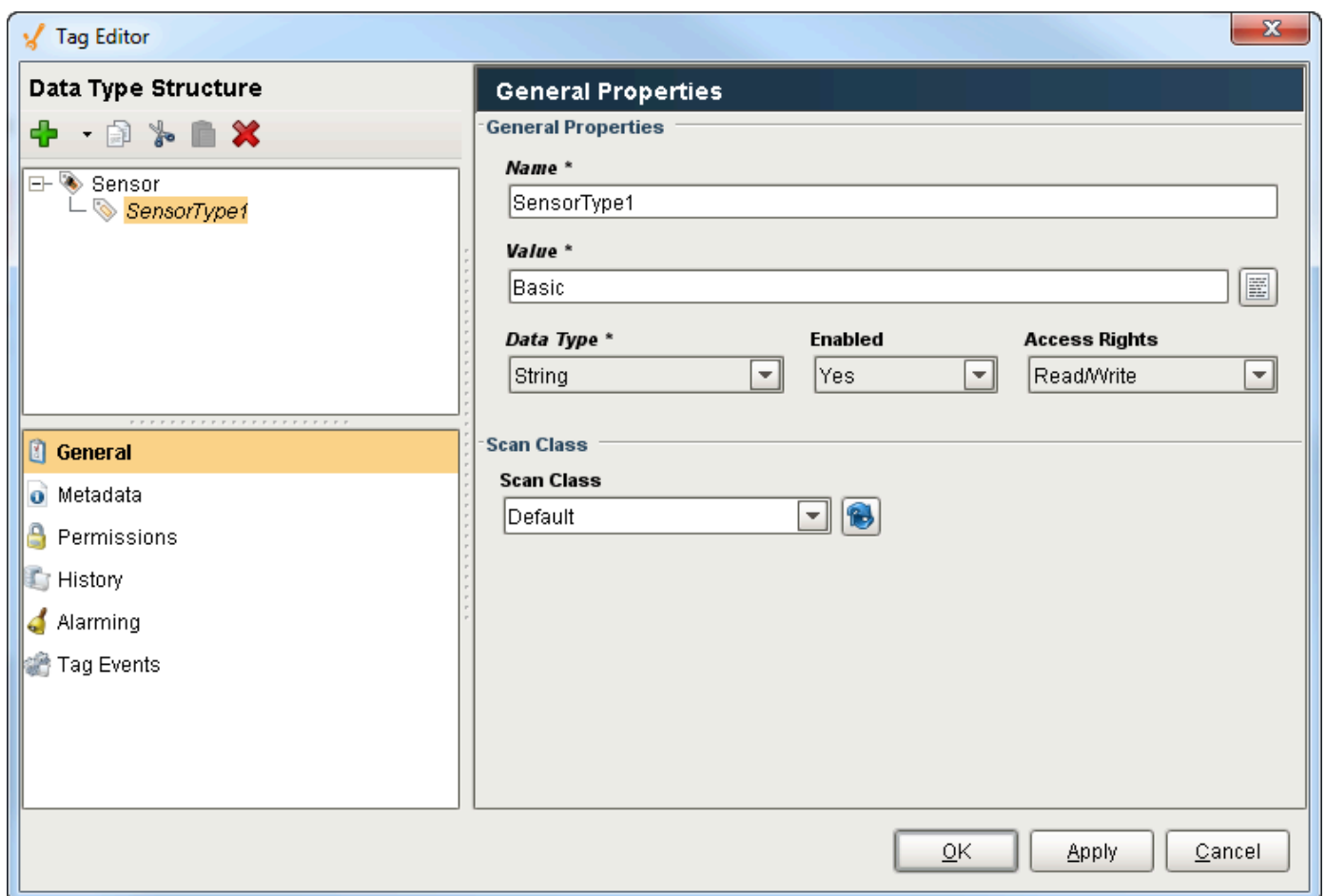
When creating a new data type manually, you have to specify the Name of the data type, parameters, and members or Tags that are going to be part of the structure of your new data type.

1. Select the **Tags** folder in the **Tag Browser**.
2. Click the **Tag** icon on the **Tag Browser** toolbar. The Tag creation menu will open.



3. Select **New Data Type** from the menu. The **Tag Editor** window will open.

When editing complex Tags, the **Tag Editor** window appears a bit differently. The member tree structure is presented in the upper left. By selecting a member, the Tag property categories are displayed below the member area, and the editor for the selected category appears to the right.



To Extend Other Types

User Defined Types can extend other UDTs to add additional members, or override default values. The **Parent Type** property (with the main/top UDT member selected) can be modified by selecting another UDT from the dropdown. This will add all members of the selected UDT to this UDT, and you

can add additional Tags or change default properties. Note: the Parent Type can only be selected when the Tag is first created. After that, it is not possible to modify the Parent Type property.

To Add Members to a Data Type

Simply click the **green +** icon at the toolbar above the member tree. Data types can contain standard Tags like OPC and DB Tags, as well as folders and instances of other complex types.

To Add Parameters

Parameters, which can be used for property expansion in member Tags, can be added by selecting the data type in the member tree. If a data type contains other complex types in it, there may be various points in the tree with custom parameters. While a data type can override the parameter values inherited from a parent, new parameters can only be added to the root node of the new data type.

Configuring Member Properties

The Tags inside of data types are configured much like normal Tags. However, in this case, the values can be thought of more as "default values", which will be used unless other values are specified when the instance is created. Most of the values configured in the data type can be modified later in subtypes or instances. Furthermore, unlike normal Tags, in the context of a data type many properties (generally the string based properties) can reference the custom attributes of the type in order to build parameterized Tags.

Attribute Referencing and Parameterized Types

As mentioned above, many properties in the member Tag configuration can reference the parameters available in the data type. When instances are created, these references are replaced with the values defined for the type. Parameter references also support basic offsets and numerical formatting, providing a great deal of flexibility. To reference a parameter, use the syntax `{ParameterName}`, or use **Ctrl-Space** to display a list of available parameters to choose from.

To offset a value, use the form `{ParameterName+offset}`. To format a value, use the form `{ParameterName|format}`. The format pattern is the same as that used for the `numberFormat` expression function. In short, "0" can be used to require a digit, and "#" can be used for optional digits. ie: **##0**

Example

For this example, we'll assume that we're parameterizing the **OPC Item Path**, and that the data type has an integer attribute named `BaseAddress` defined. We'll pretend the **OPC Server** provides Tags named like `DataPoint1`.

Standard Referencing

OPC Item Path: `DataPoint{BaseAddress}`

Offset

Imagine that our data type had three fields, and these were laid out sequentially in the device. Instead of specifying each address for each Tag, we can simply offset from the base address:

Member 1: `DataPoint{BaseAddress+0}`

Member 2: `DataPoint{BaseAddress+1}`

Member 3: `DataPoint{BaseAddress+2}`

Formatting (with Offset)

Continuing from the example above, imagine that our OPC server actually provided addresses in the form `DataPoint001`, in order to stay consistent up to "DataPoint999". This can be accommodated using number formatting in the reference:

Member 1: `DataPoint{BaseAddress+0|000}`

Member 2: `DataPoint{BaseAddress+1|000}`

Member 3: `DataPoint{BaseAddress+2|000}`

This format of three zeros means "three required digits". If our instance has a base address of 98, the resulting paths will be `DataPoint098`, `DataPoint099`, `DataPoint100`.

Properties that can be Parameterized

The following Tag properties can reference parameters:

- Value (for string data type only)
- OPC Server
- OPC Item Path
- Tooltip
- Documentation
- Expression/SQL Query

- Bindable Alarm Properties (Note: you can bind a property directly to the parameter, or use parameters in the binding expression, or directly in string property values)

CAUTION

Using parameters in expressions is a little different than using properties in an expression. These parameters are pre-processed, meaning they are evaluated before the expression is run. So if they are a string and you want to concatenate them, you must have them inside quotes.

Data Type	Value	Expression	Result
number	2	{motorNumber}+2	4
string	M2	{motorName}+" Title"	ERROR
string	M2	"{motorName} Title"	"M2 Title"

Overriding Properties

Subtypes and instances can override the properties defined in parent types. To do this, simply select the **override control** (the small grey ball) next to the property to override in the member editor. Conversely, to remove the override, simply unselect the control.

Custom parameters can be overridden as well, but it is not required to specify that the value is an override. Simply provide a new value for the property. For inherited parameters, the **delete** button next to the parameter table will simply remove the override. The parameter can only truly be delete from the type that defines it.

Pre-Defined Parameters

UDTs have a few parameters already defined to make things easier for you. They give you access to the name and various paths associated with a UDT member Tag. These parameters can be accessed from anywhere in a Tag that a normal parameter can be used. Each of these parameters uses that Tag it is in as a starting point for it's path.

Parameter Name	Description
{InstanceName}	The name of the UDT Instance that this Tag is inside.
{PathToParentFolder}	The full path to the folder that this Tag is in.
{PathToTag}	The full path to the Tag using this parameter.
{TagName}	The name of the Tag that is using this parameter.

Tag Browser		
Tag	Value	Data Type
Tags		
Data Types		
InstFolder		
UDTInstance		ShowParams
Folder		
InstanceName	UDTInstance	String
PathToParentFolder	InstFolder/UDTInstance/Folder	String
PathToTag	InstFolder/UDTInstance/Folder/PathToTag	String
RootPath	InstFolder/UDTInstance	String
TagName	TagName	String

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Creating UDT Instances



INDUCTIVE
UNIVERSITY

Creating UDT Instances

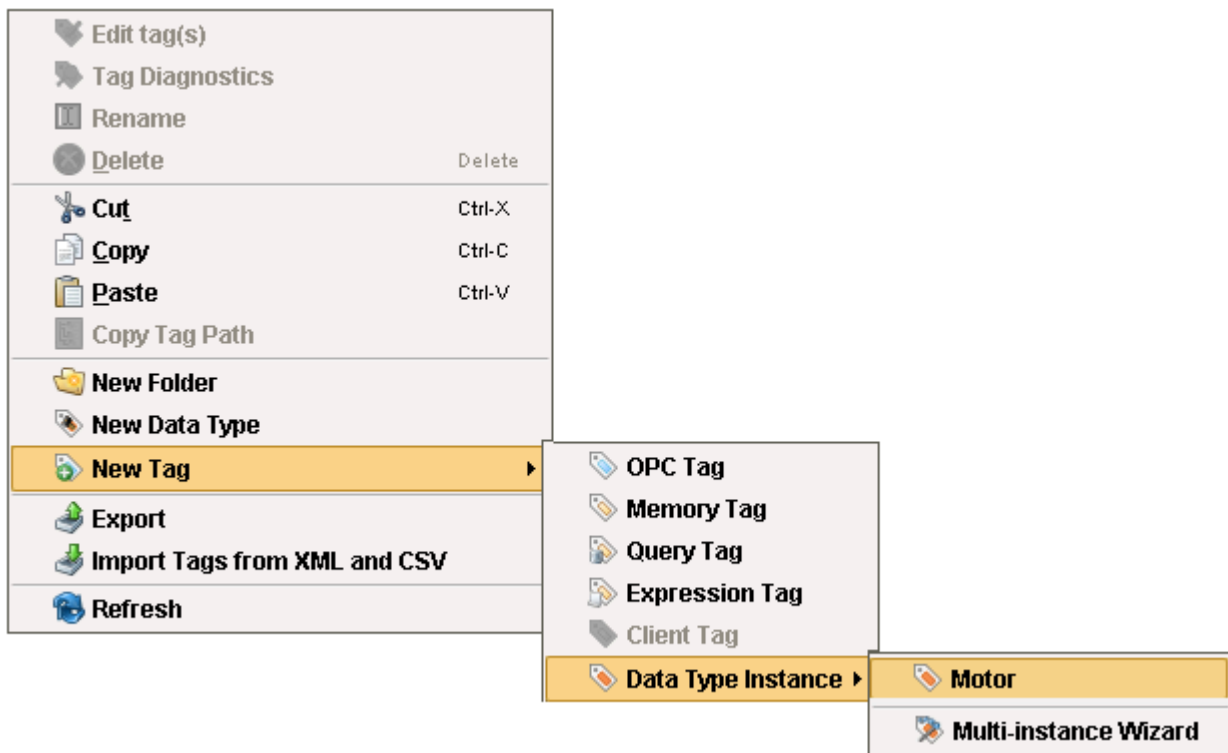
[Watch the video](#)

Creating instances of UDTs is virtually identical to creating other types of Tags using the **New Tag** menu. Unlike standard Tags, it is likely that you'll have to modify attribute values or override certain member properties in order to make the instance unique.

To Create a UDT Instance

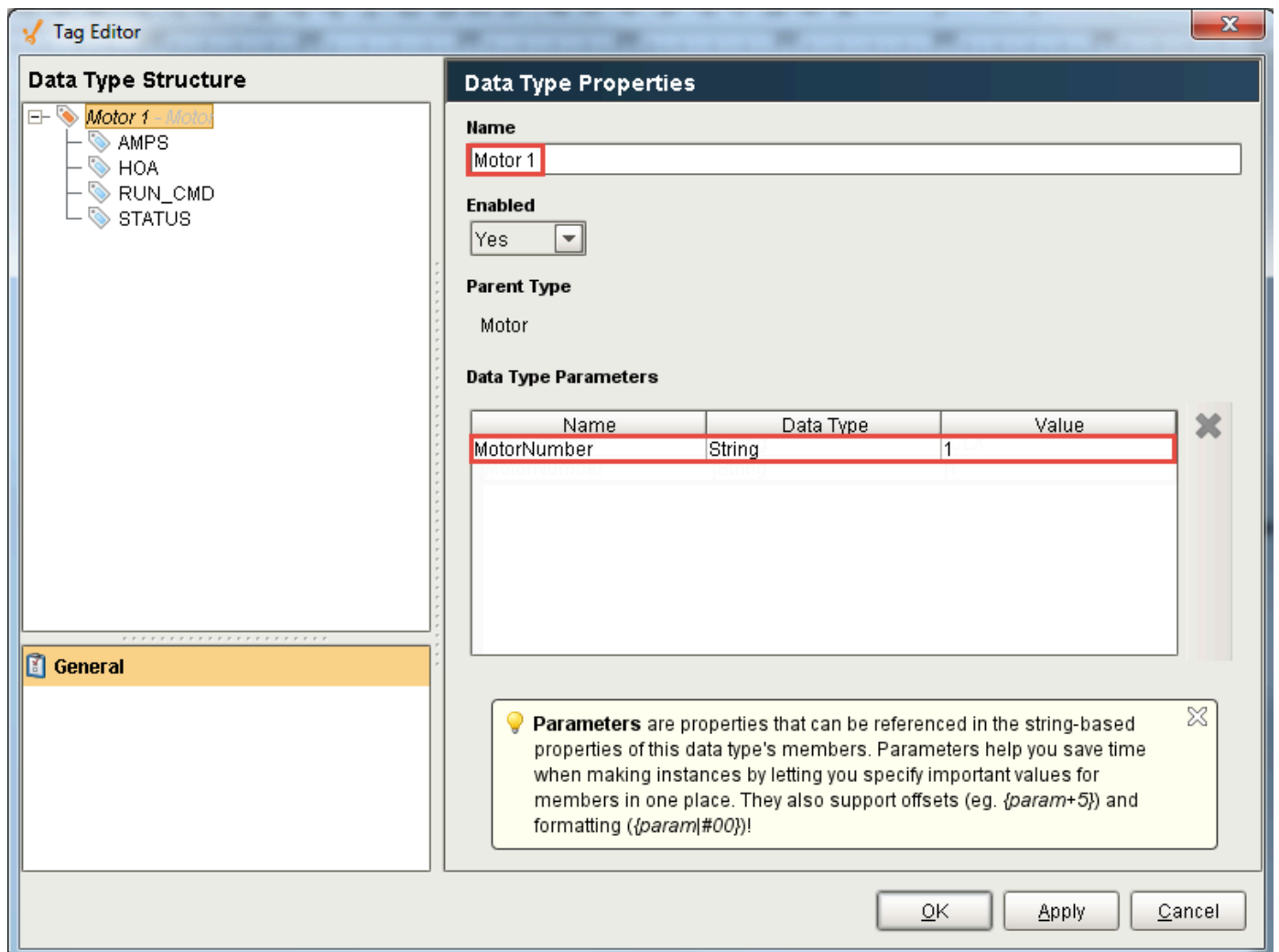
Once a **UDT definition** is created, you can create an instance of the UDT as an actual Tag in Ignition. Now that you have the Motor UDT created from the previous section, let's create a UDT instance.

1. In the **Tag Browser**, right-click on **Tags** and select **New Tag > Data Type Instance > Motor** to create a new instance.

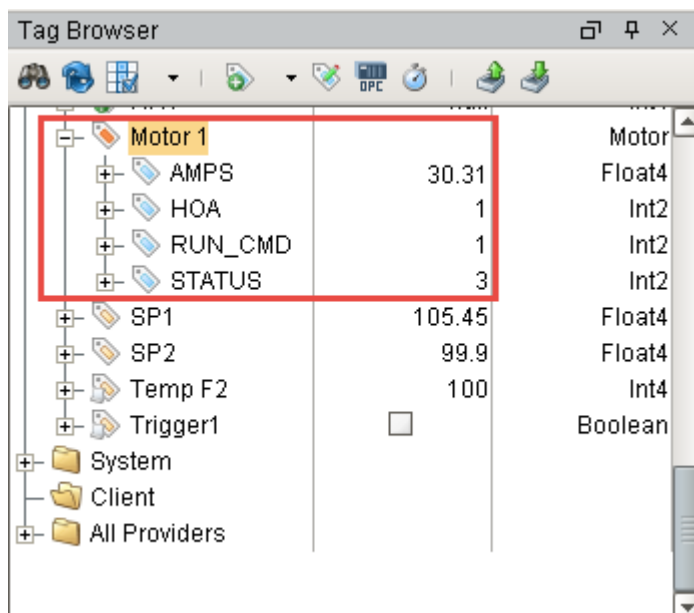


2. Assign the new instance a name, **Motor 1** and enter a value for the MotorNumber parameter that is used in the configuration of the UDT Motor.

- Assign the **MotorNumber** parameter a value of **1** for **Motor 1**.
- Click **OK**, and the Motor 1 instance is created in Ignition.

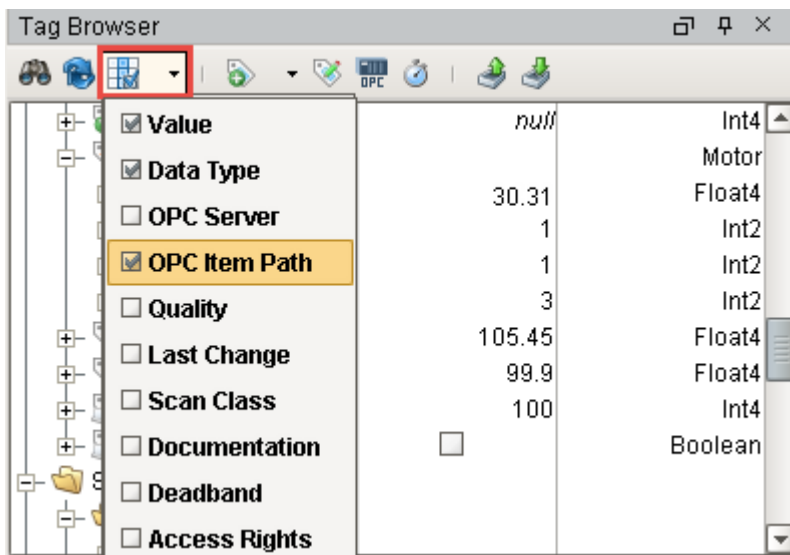


3. From the **Tag Browser**, expand **Motor 1** to verify that the Tags are working correctly.

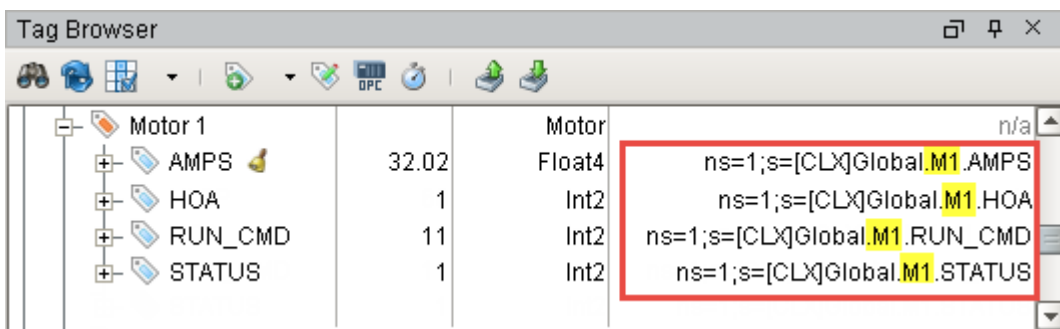


4. In the **Tag Browser**, check the **OPC Item Path** to see how Ignition translated the path for the MotorNumber parameter, and verify that it is correct and of good quality.

- From the **Tag Browser**, select **Motor 1**, click the **Column Selector**, and check **OPC Item Path** from the dropdown.

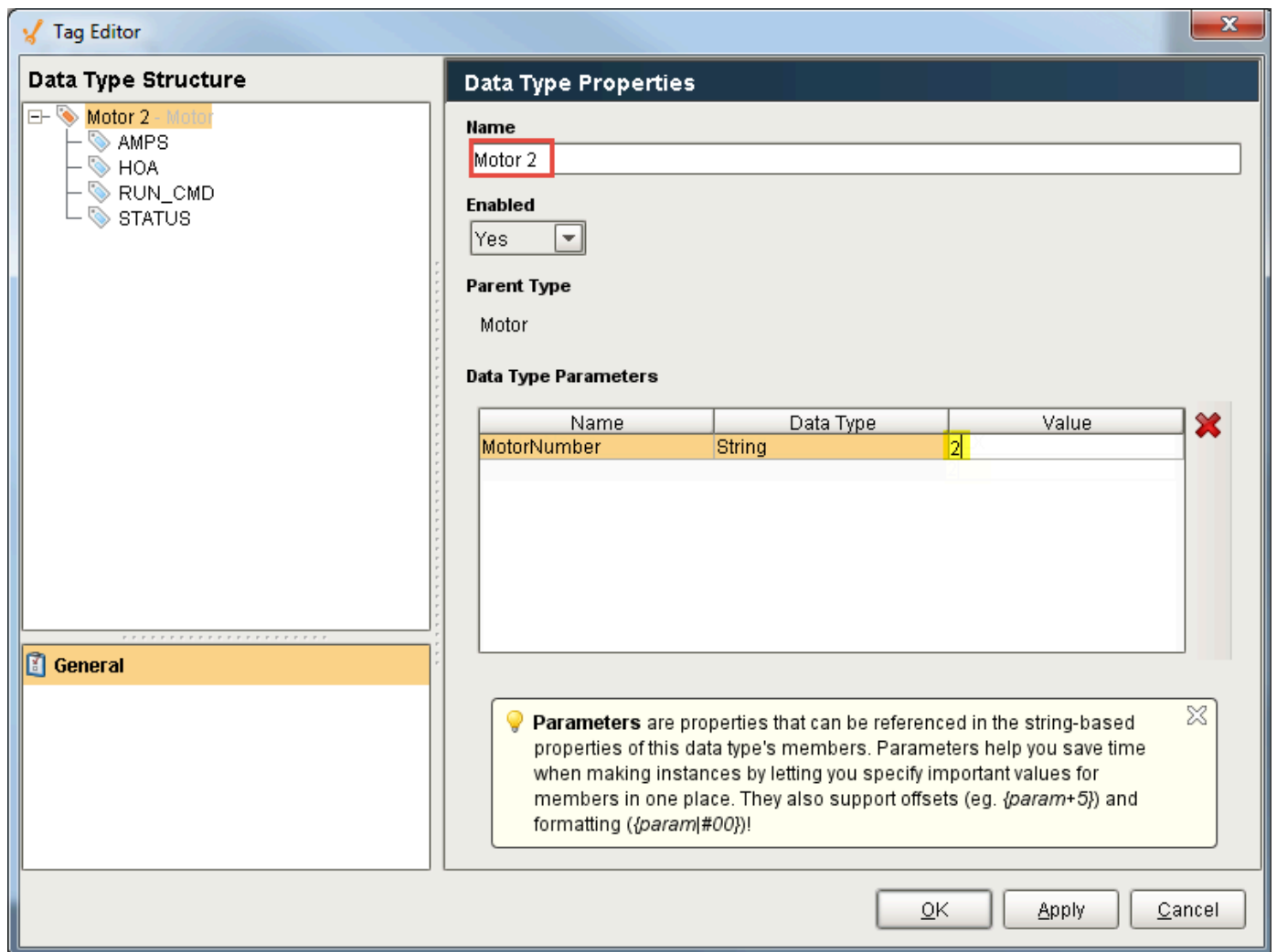


You can see that Ignition displays the **MotorNumber (M1)** correctly.



5. To create another instance of Motor 1, **copy** and **paste Motor 1** in the Tag folder to create **Motor 2**.

- Open the **Tag Editor** and modify the **MotorNumber** value from **1** to **2** to point to a different Motor (i.e., Motor 2).
- Click **OK**.



6. To confirm Motor 2 is working correctly, open the **Tag Browser** and expand **Motor 2**.

The screenshot shows the 'Tag Browser' window. It displays a tree view of tags. 'Motor 1' is expanded, showing 'AMPS' (30.67, Float4), 'HI SP' (60, Int4), 'HOA' (1, Int2), 'RUN_CMD' (11, Int2), and 'STATUS' (1, Int2). 'Motor 2' is also expanded, showing 'AMPS' (7.41, Float4), 'HOA' (1, Int2), 'RUN_CMD' (2, Int2), and 'STATUS' (2, Int2). The right pane shows the path for each tag, with the paths for Motor 2's members highlighted in a red box.

Tag Name	Value	Data Type	Path
Motor 1		Motor	n/a
Motor 1.AMPS	30.67	Float4	ns=1;s=[CLX]Global.M1.AMPS
Motor 1.HI SP	60	Int4	n/a
Motor 1.HOA	1	Int2	ns=1;s=[CLX]Global.M1.HOA
Motor 1.RUN_CMD	11	Int2	ns=1;s=[CLX]Global.M1.RUN_CMD
Motor 1.STATUS	1	Int2	ns=1;s=[CLX]Global.M1.STATUS
Motor 2		Motor	n/a
Motor 2.AMPS	7.41	Float4	ns=1;s=[CLX]Global.M2.AMPS
Motor 2.HOA	1	Int2	ns=1;s=[CLX]Global.M2.HOA
Motor 2.RUN_CMD	2	Int2	ns=1;s=[CLX]Global.M2.RUN_CMD
Motor 2.STATUS	2	Int2	ns=1;s=[CLX]Global.M2.STATUS

Creating UDT Instances is really simple once you have a UDT definition in place.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

UDT Multi-Instance Wizard



INDUCTIVE
UNIVERSITY

UDT Multi-Instance Wizard

[Watch the video](#)

The multi-instance wizard provides a powerful, but simple mechanism for rapidly generating many instances of a UDT at the same time by specifying patterns for UDT parameters.

Value Patterns and Tag Names

Value Patterns

In order to define values for parameters (and the Tag names), you can use several different types of patterns (and combinations of patterns):

Name	Value	Description
Range	<code>number1- number2[/step]</code>	A numeric range of values, such as 1-10. Optionally, a <code>step</code> parameter can be included, in order to only generate numbers at certain multiples. For example, 0-100/10 would generate 0,10,20, and so on.
Repeat	<code>value*count</code>	A value (numerical or string), and the number of times to use it. For example, <code>North Area*10</code> would use the parameter <code>North Area</code> for 10 items.
List	<code>value1, value2, value3</code>	A comma separated list of values (or other patterns) to use.

Examples:

1-10,20-30,30-40	Results in 30 Tags being created, with the specified value ranges (so, for example, there would be no parameter 15).
A,B,C	Results in 3 Tags, with each of the values.
0-100/5	Results in 21 Tags (because range is inclusive), with values 0, 5, 10...100.

As mentioned, the size of the pattern will dictate how many Tags will be created. If some patterns are smaller than others, the last value will be repeated for the other Tags.

Tag Names

The names of the generated instances can be specified using a system similar to that of the parameter patterns. If you just want to use sequential names, you don't need to specify a pattern, as values will be generated automatically starting at one. You can also set the pattern to simply be the starting number to generate sequential names from there.

Base Name

A string base for the Tag name. This can also be a list of names, in which case the names will be used directly, and the name pattern won't be used.

Name Pattern

A pattern that will be used to generate values that will be appended to the base name.

At any time, you can use the **Preview** button to view the Tag names and parameters that will be created. Once you are satisfied, click **OK** to generate the Tags under the selected folder in the Tag provider.

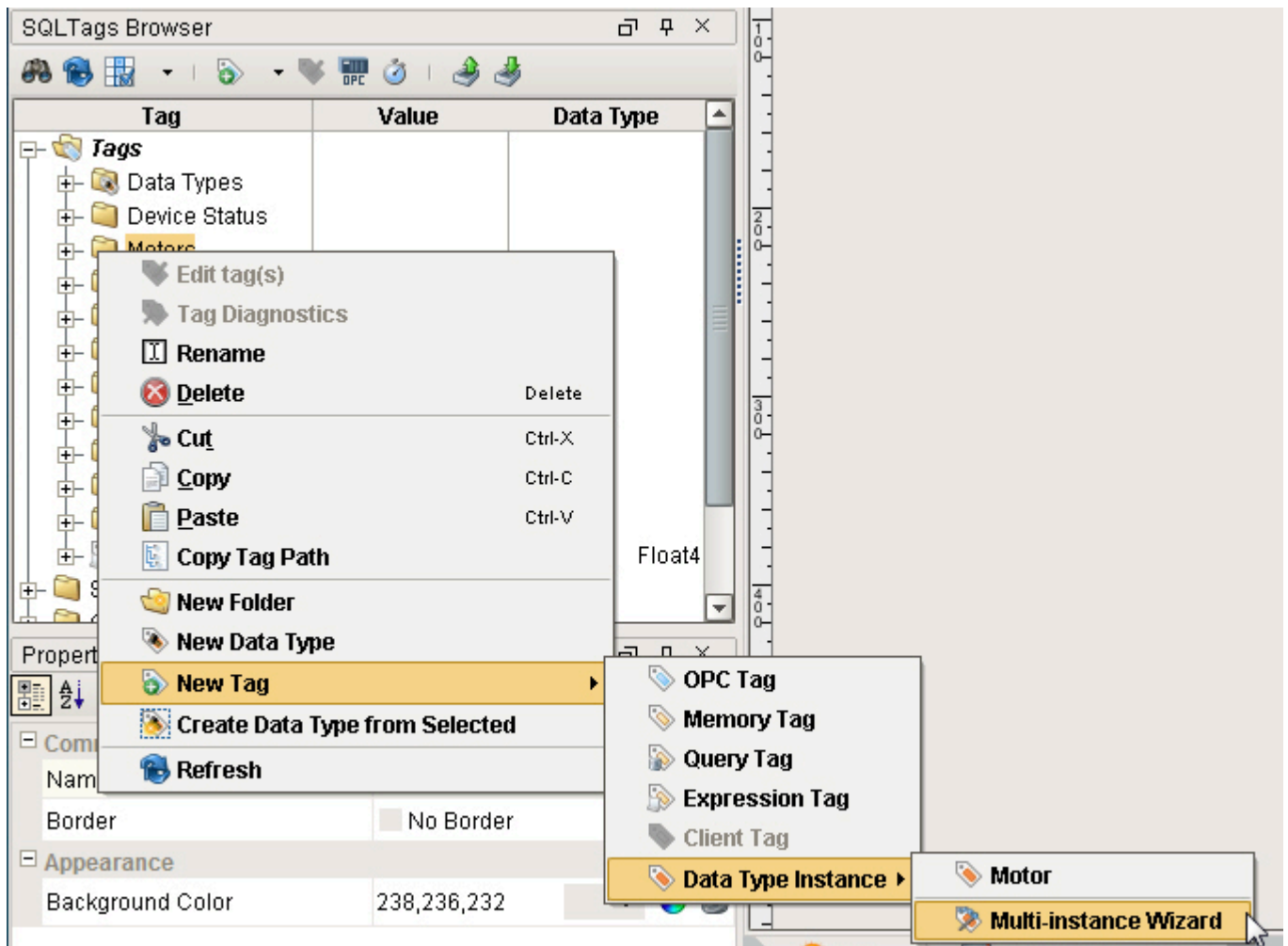
How to Make New Instances of a UDT

Now that we have the Motor UDT (see [Creating User Defined Types - UDTs](#)) we can make instances of it. Instances are running copies of a data type with concrete data for all members.

1. In the **Tags Browser**, right-click on **Tags** and select **New Folder** to create a new folder called **Motors**.

You can create a single instance of the motor or use the Multi-Instance Wizard to rapidly create all four at the same time. In this example, we will use the Multi-Instance Wizard.

2. Right-click on the **Motors** folder, and select **New Tag > Data Type Instance > Multi-Instance Wizard**.



3. The Instance Creation Wizard is displayed. From the **Step 1 - Select Data Type to Create**, select **Motor** from the dropdown list.
4. Configure the following parameters:
 - **Base Tag Name:** Motor
 - **Tag Name Pattern:** 1-4 (This creates four Tags: Motor 1, Motor 2, Motor 3, Motor 4.)
 - **Parameter:** MotorNumber (This is used to reference specific Tags in the PLC.)
 - **Pattern:** 1-4 (This assign Motor 1 Tag a parameter of 1, Motor 2 a parameter of 2, and so on.)

Instance Creation Wizard

Step 1 - Select Data Type to Create

Motor

Step 2 - Configure Parameters

Base Tag Name: Motor **Tag Name Pattern:** 1-4

Parameter Patterns

Parameter	Type	Pattern	Size
MotorNumber	String	1-4	4

Tags to create: 4

Note: The pattern is used to generate values, and can be:
Range: *x-y[/step]*, for numerical values. Example: 0-10, 0-100/10
List: *a,b,c* Example: West,East,North,South
Repeat: *value*count* Example: 1*10
 List patterns can also contain range and repeat patterns as values. For example, 0-10,20-30,40*10

Preview OK Cancel

5. Click **Preview** to see how Ignition creates the Tag name and the Parameter Value.

Tag Name	MotorNumber
Motor 1	1
Motor 2	2
Motor 3	3
Motor 4	4

6. Click the **Back** button to exit the Preview screen to go back to the Instance Creation Wizard screen.

Click **OK** to create your instances. Now, expand all your Motor Tags to see if all the members of the UDT were created and are running.

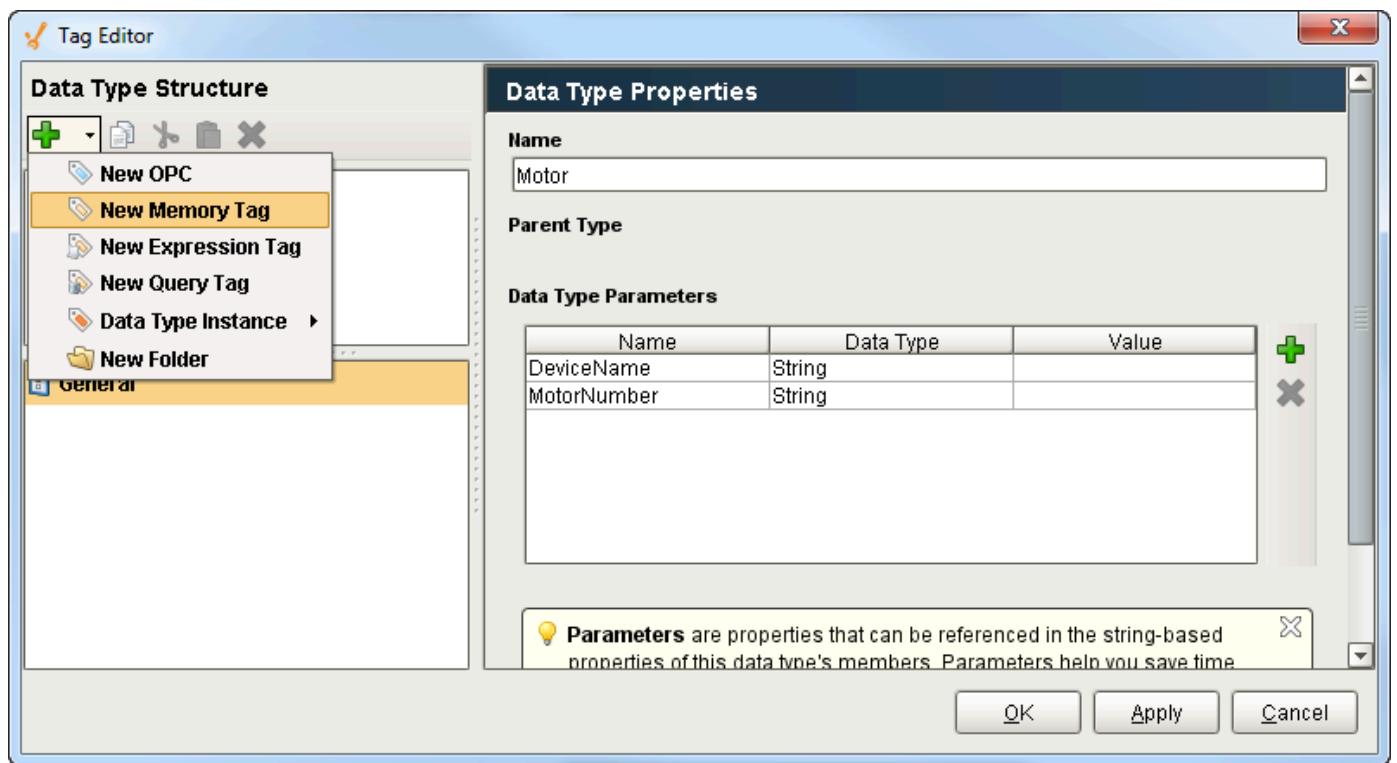
The screenshot shows the SQLTags Browser window. The left pane displays a tree structure under 'Tags'. The right pane shows a table with three columns: 'Tag', 'Value', and 'Data Type'.

Tag	Value	Data Type
Tags		
Data Types		
Motor		
Device Status		
Motors		
Motor 1		Motor
AMPS	45	Float4
HOA	1	Int2
RUN_CMD	0	Int2
STATUS	0	Int2
Motor 2		Motor
AMPS	20	Float4
HOA	1	Int2
RUN_CMD	1	Int2
STATUS	0	Int2
Motor 3		Motor
Motor 4		Motor
Motors 1		
Overview		

How to Edit an Existing UDT

If you make a change to a UDT, all your instances will automatically get updated because they are all of the same UDT type.

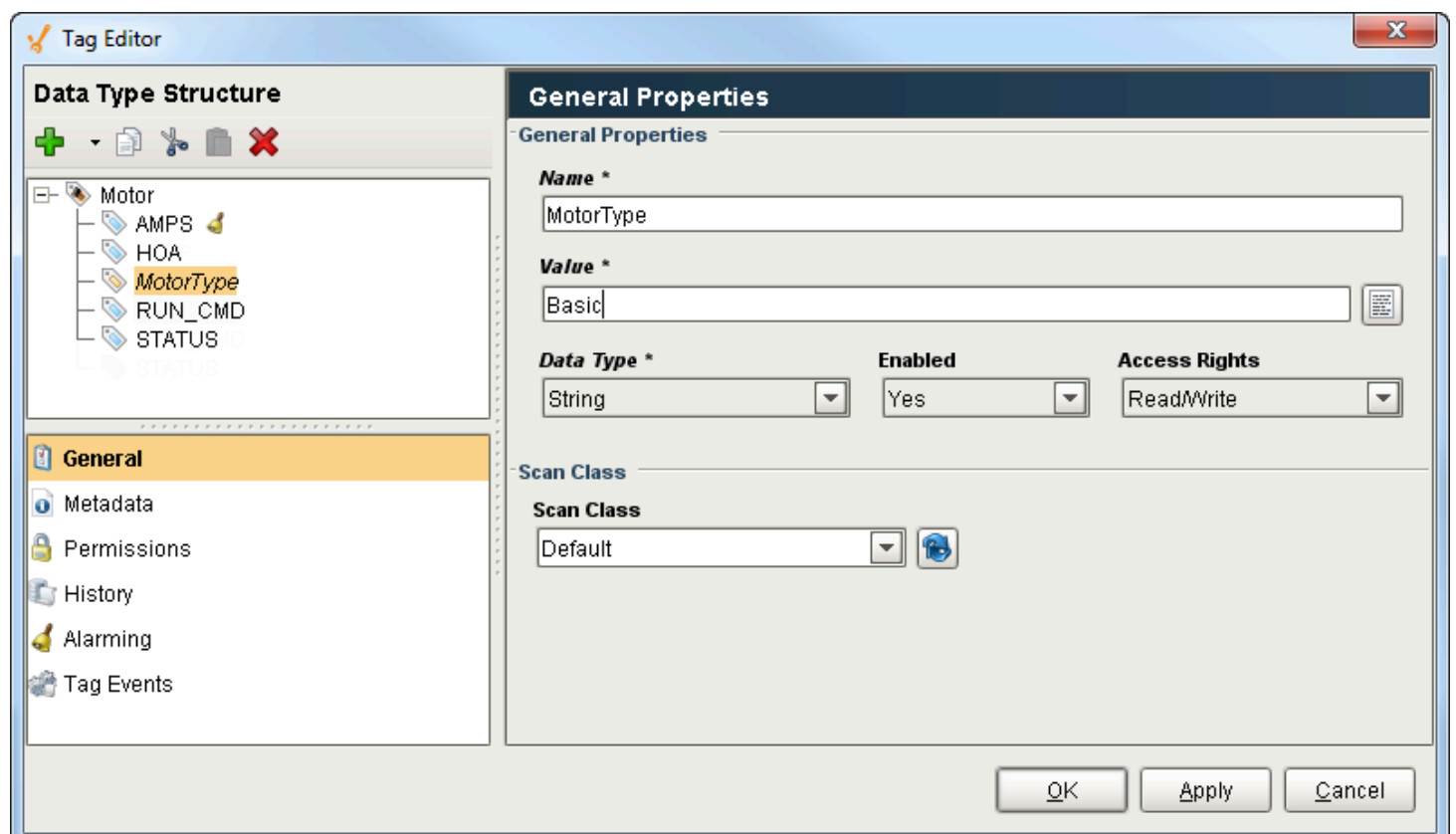
1. To make a change to your Motor UDT, go to the Tag Browser and double click on Motor. This example adds a new Memory Tag.
2. The Tag Editor will open. Click on the green + button on the top-left of window, and select New Memory Tag.



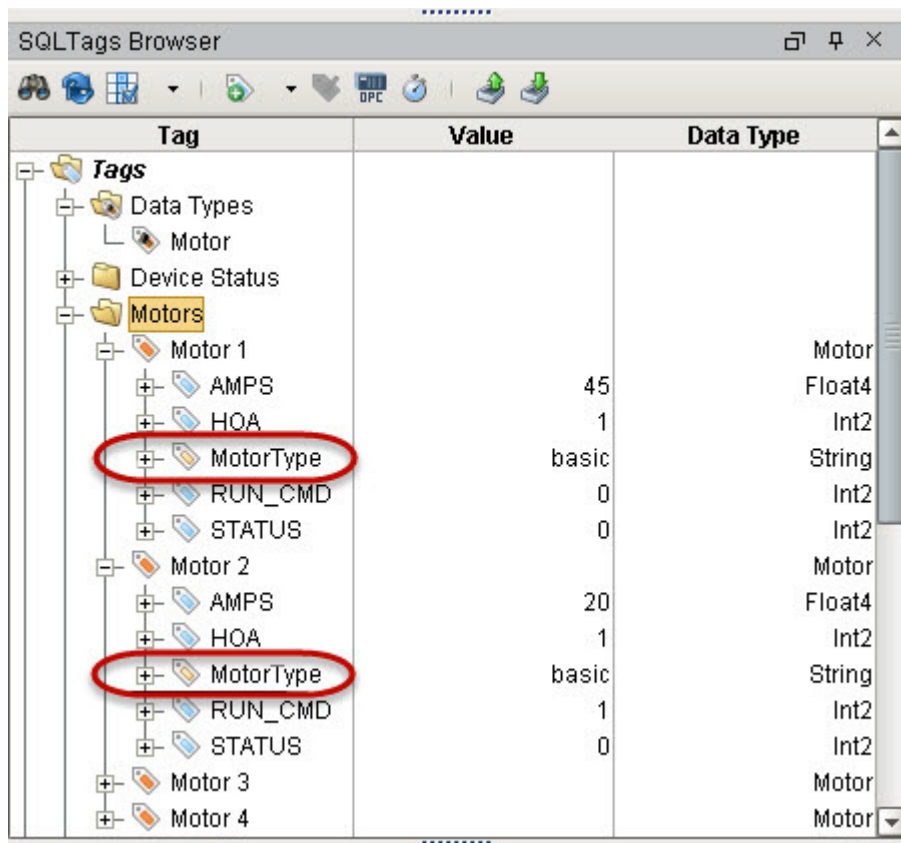
3. Configure the following properties:

- **Name:** MotorType
- **Data Type:** String
- **Value:** Basic

Click **OK**.



4. Now, each instance automatically gets the new Memory Tag. Expand each of your Motor Tags to see the new MotorType.



Tag	Value	Data Type
Tags		
Data Types		
Motor		
Device Status		
Motors		
Motor 1		Motor
AMPS	45	Float4
HOA	1	Int2
MotorType	basic	String
RUN_CMD	0	Int2
STATUS	0	Int2
Motor 2		Motor
AMPS	20	Float4
HOA	1	Int2
MotorType	basic	String
RUN_CMD	1	Int2
STATUS	0	Int2
Motor 3		Motor
Motor 4		Motor

You can make rapid work of making changes to instances using a UDT.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Overriding Properties in UDT Instances



INDUCTIVE
UNIVERSITY

Overriding Properties in UDT Instances

[Watch the video](#)

It is possible to override properties of a UDT instance to create a different base structure as compared to other instances of the same data type. This is an important feature in certain scenarios where you have to be able to override parts of the configuration of that instance. For example, you may have to create an instance of a UDT that uses a completely different OPC address scheme from the definition.

To Override the Properties

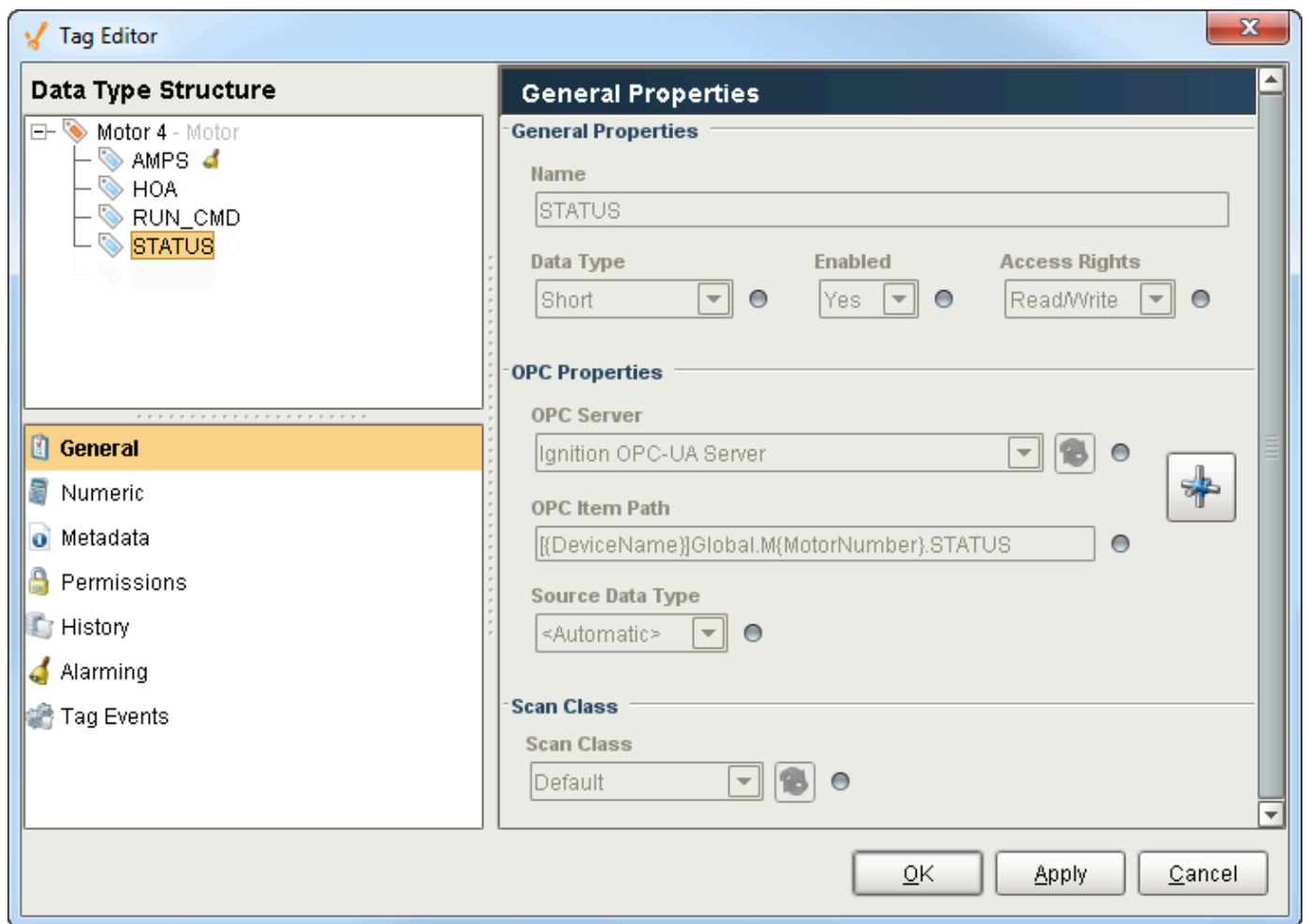
All the instances of the same data type definition have the same configuration, but when there is an instance which needs to have a different value from the rest of them, you can override that value. Let's say we want to override the **OPC Item Path** property of the **STATUS** Tag of the **Motor 4** instance.

1. From the **Tag Browser**, you would go to **Tags > Data Types**, right-click on **Motor 4**, then select **Edit tag(s)**.

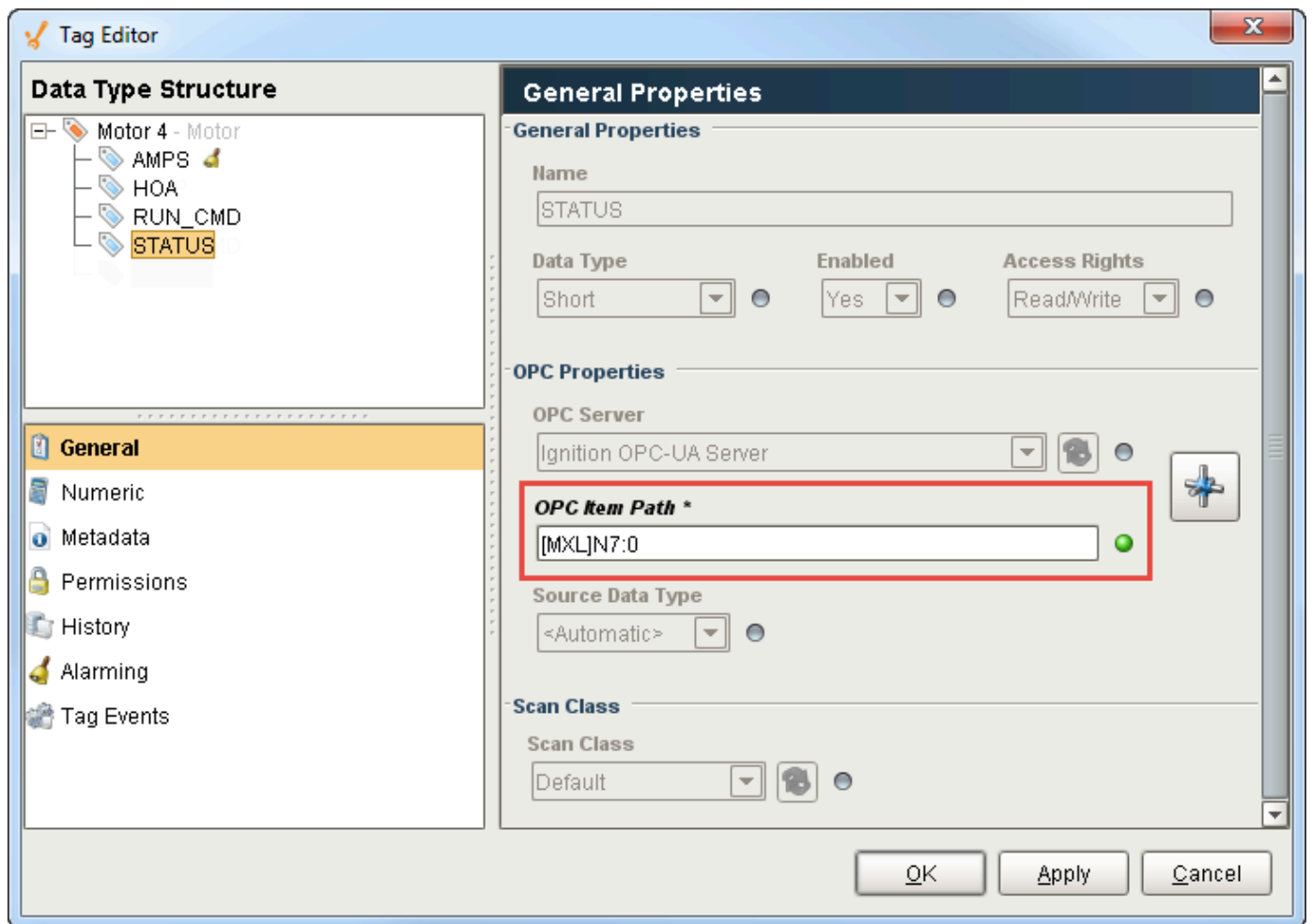
The **Tag Editor** window will open.

2. Under **Data Type Structure**, select the **STATUS** Tag.

You can see that none of the fields are enabled so you can't make any changes to them.



3. Look for the circle to the right of the **OPC Item Path** box and click it so you can highlight this box. The circle changes its color to green and the field becomes enabled allowing you to override the OPC Item Path inherited from the parent.
4. Edit the **OPC Item Path**, and click **OK**.



5. Go to the **Tag Browser**, click on the **Column selection** icon on the toolbar, and select the box for **OPC Item Path**.

See your changes in the column that displays the value of the OPC Item Path.

Tag Name	Value	Data Type	OPC Item Path
Motor 1		Motor	n/a
Motor 2		Motor	n/a
Motor 3		Motor	n/a
Motor 4		Motor	n/a
AMPS	43	Float4	[CLX]Global.M4...
HOA	2	Int2	[CLX]Global.M4...
RUN_CMD	1	Int2	[CLX]Global.M4...
STATUS	2	Int2	[MXL]N7:0

6. To go back to the original configuration, you can unclick the green circle next to the **OPC Item Path**.

⚠ IF YOU CHANGE THE UDT DEFINITION ON A FIELD THAT WAS OVERWRITTEN

It's important to note that if you make a change to the definition on a particular field that was overwritten, it will not update on that particular instance. That particular instance is locked in at

that override unless you go back to your Tag and remove the override by clicking the green circle.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

UDT Inheritance



INDUCTIVE
UNIVERSITY

UDT Inheritance

[Watch the video](#)

It is possible to set up UDT inheritance where data types extend to other data types, to add additional members, or override default values.

To Inherit Property Values from an Existing UDT

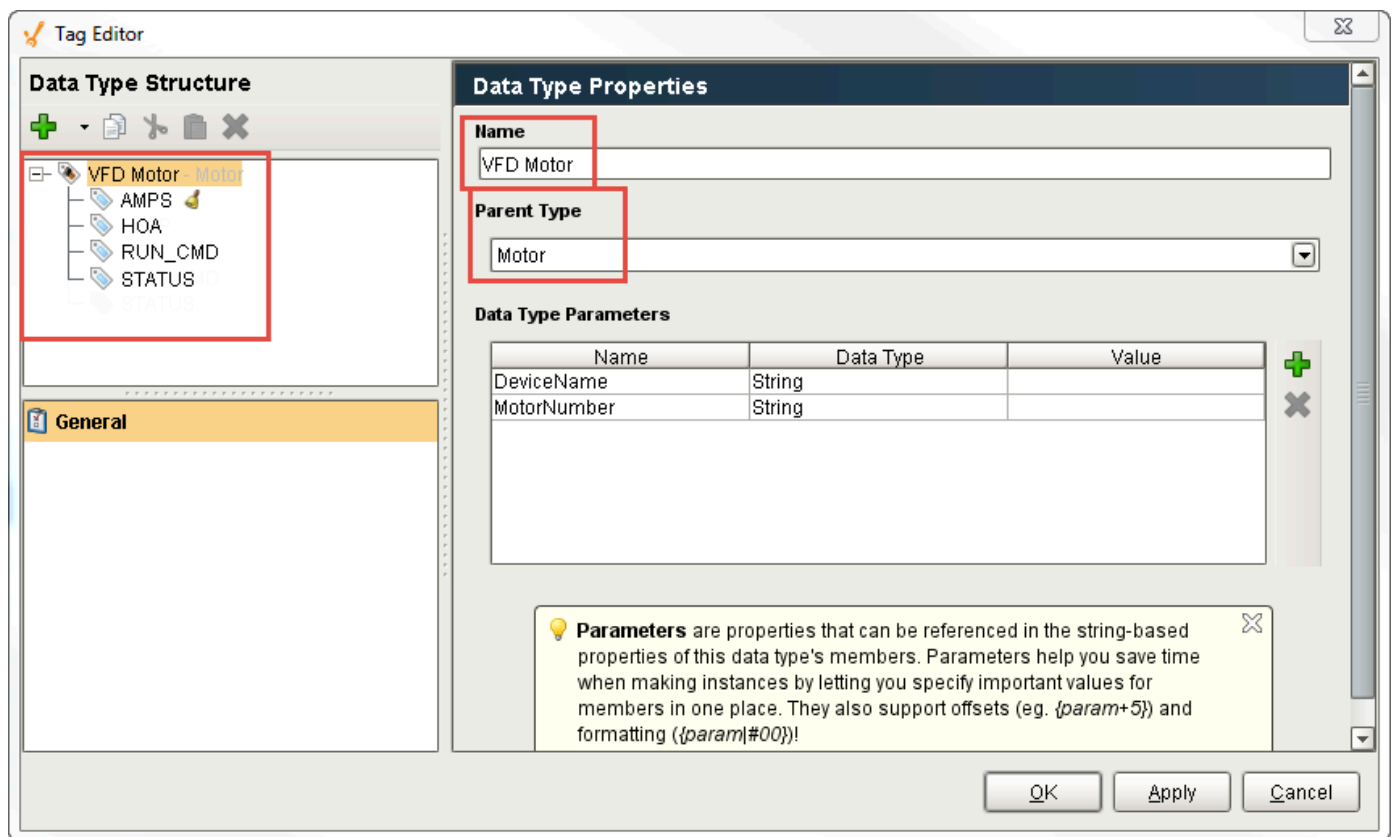
Let's use our data type **Motor**, from the previous sections, to create another data type. We'll set the parent to **Motor** so our new data type automatically inherits all the properties of **Motor**.

1. From **Tag Browser**, right-click on **Data Type**, and choose **New Data Type**.

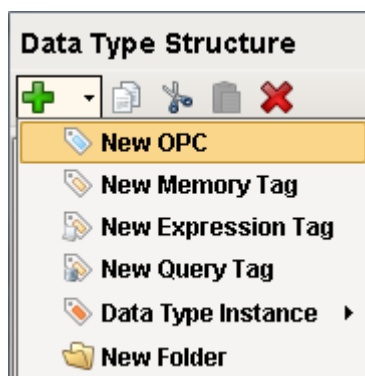
The Tag Editor window will open.

2. In the **Name** box, enter a new name (i.e. **VFD Motor**), and from the dropdown of the **Parent Type** box, choose an existing UDT (i.e. **Motor**).

This new UDT will automatically inherit all the properties of the parent UDT **Motor**: **AMPS**, **HOA**, **RUN_CMD**, and **STATUS**.



3. You can add a new Tag to this new UDT by clicking the **green + icon** on the top-left of the **Tag Editor** window, and click **New OPC**.



Enter the following properties:

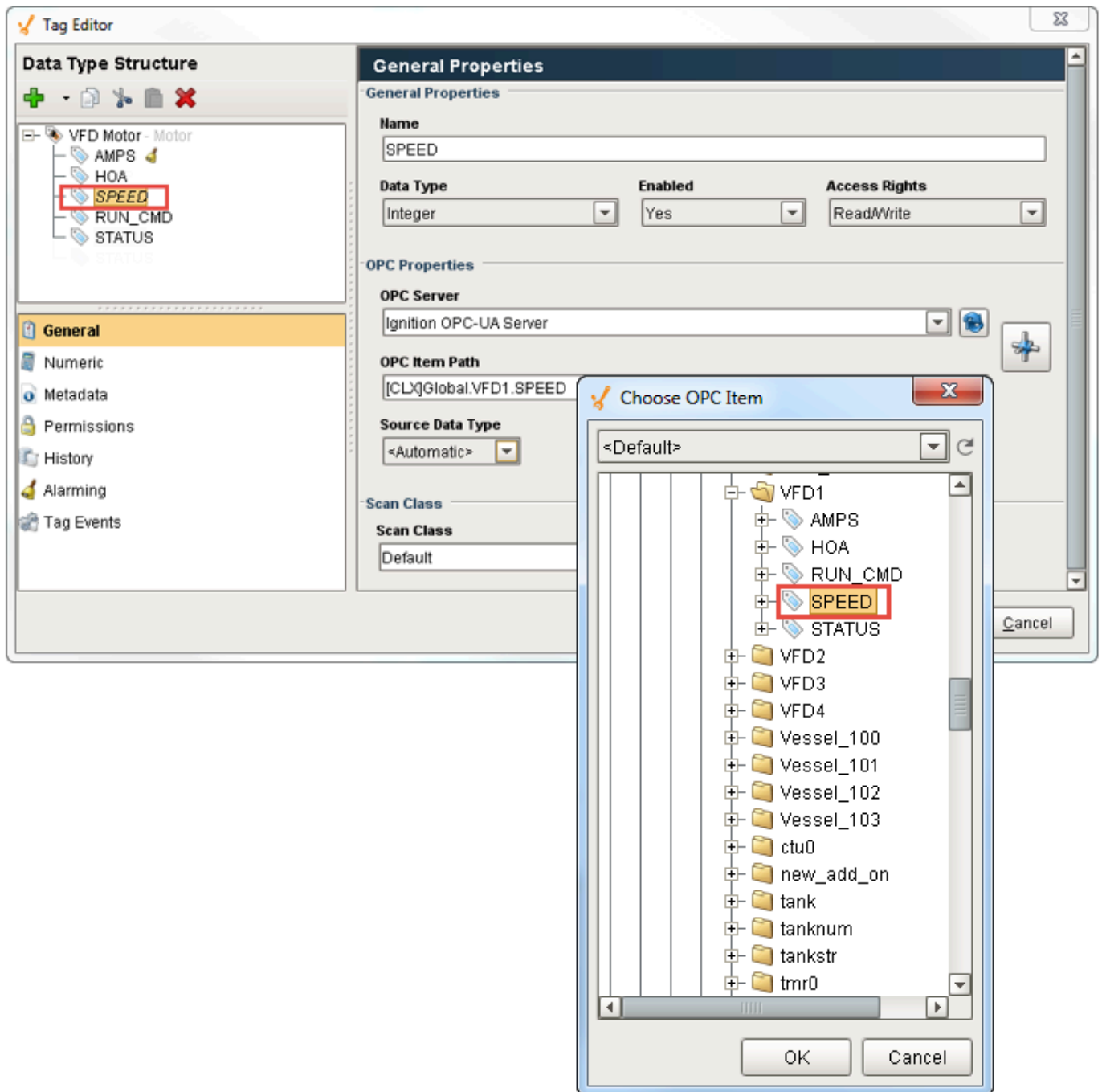
Name: **SPEED**

Data Type: **Integer**

OPC Server: **Ignition OPC-UA Server (from the dropdown list)**

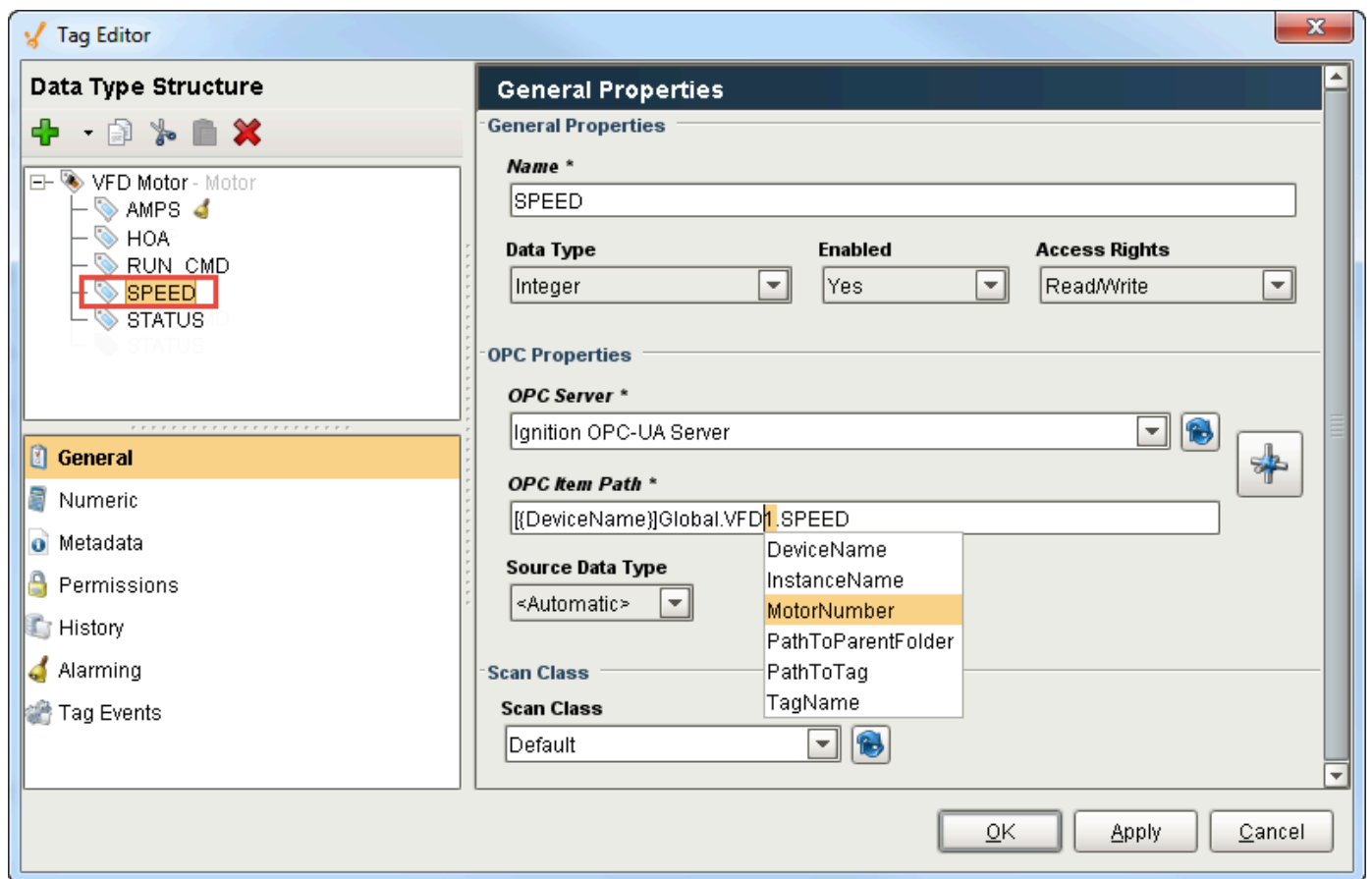
OPC Item Path: **Browse the PLC and find the VFD inside the ControlLogix program.**

Click **OK**.

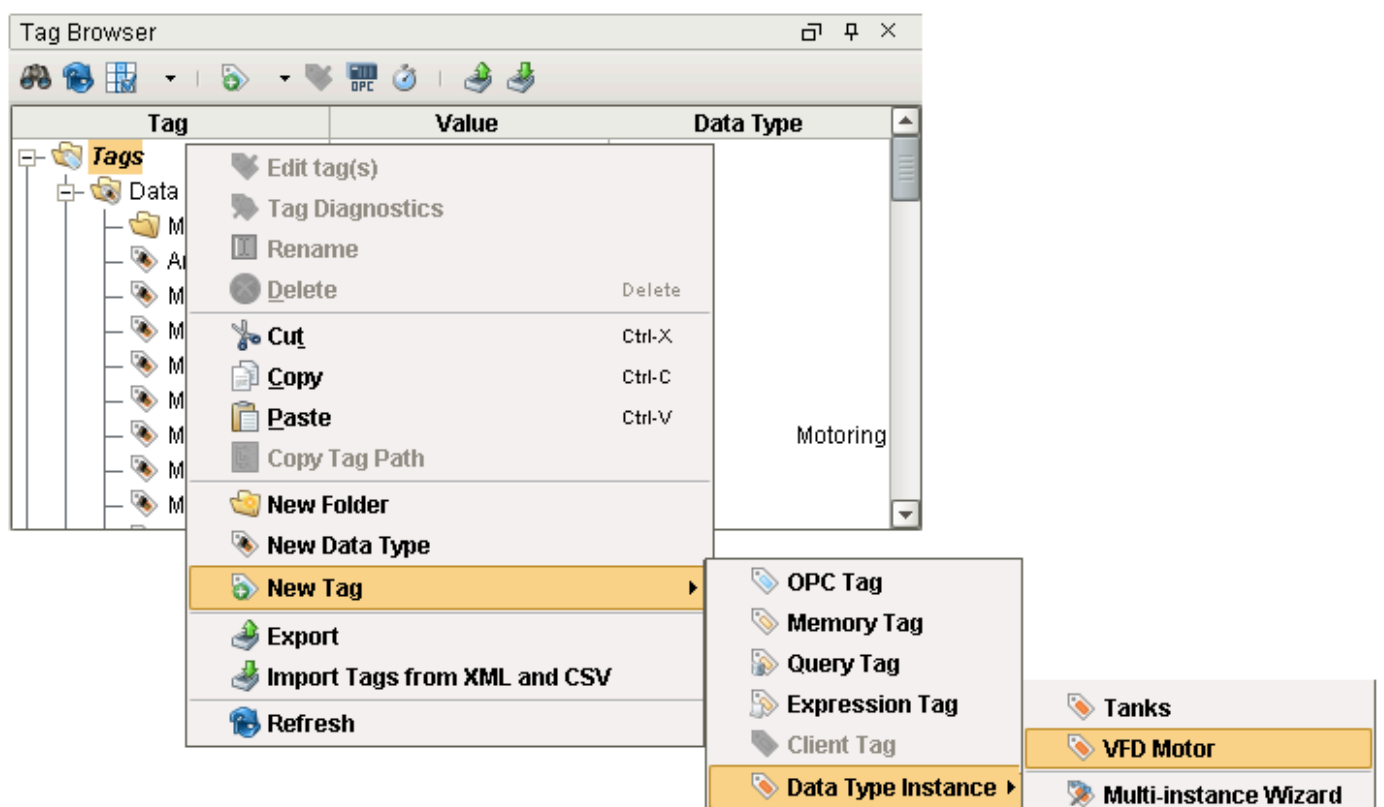


Because you are creating a UDT, you don't want to point to one specific set of Tags. You want each instance of the VFD Motor UDT to reference a different set of Tags. To do that, you need to add a parameter to the UDT.

In the **OPC Item Path**, replace the "CLX" with `{DeviceName}` using the down arrow, and the "1" with `{MotorNumber}` using the down arrow as shown below:



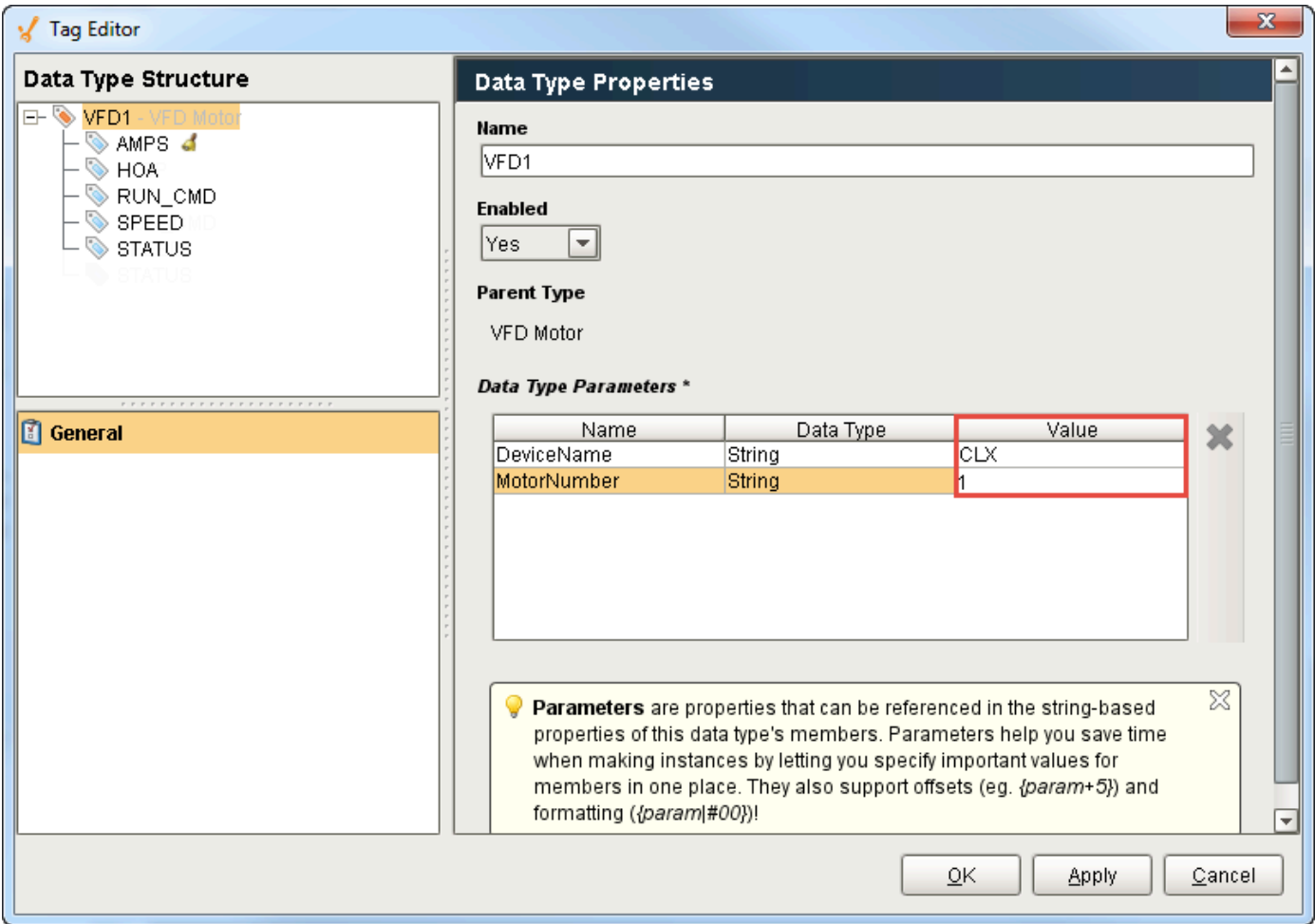
4. Click **OK** again. The new data type called VFD Motor inherited all the Tags from the Motor data type and is visible in the Tag Browser.
5. Now, create an instance of the VFD Motor. Go to the **Tag Browser**, and select **Tags > New Tag > Data Type Instance > VFD Motor**.



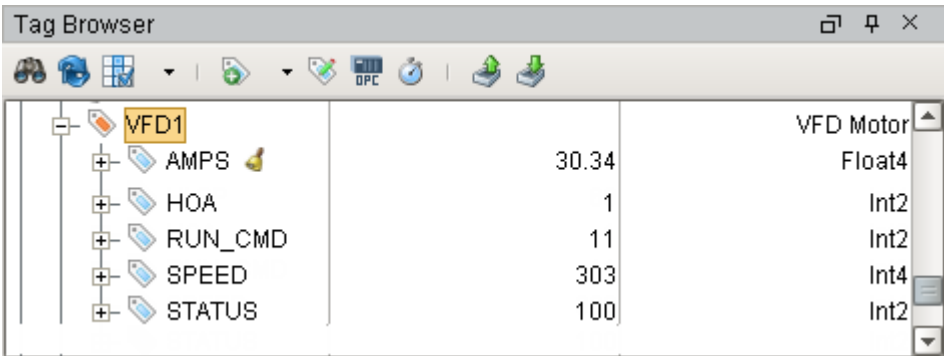
6. The Tag Editor window will open. Enter the following:

- Name: **VFD1**
- DeviceName Value: **CLX**
- MotorNumber Value: **1**

Click **OK**.

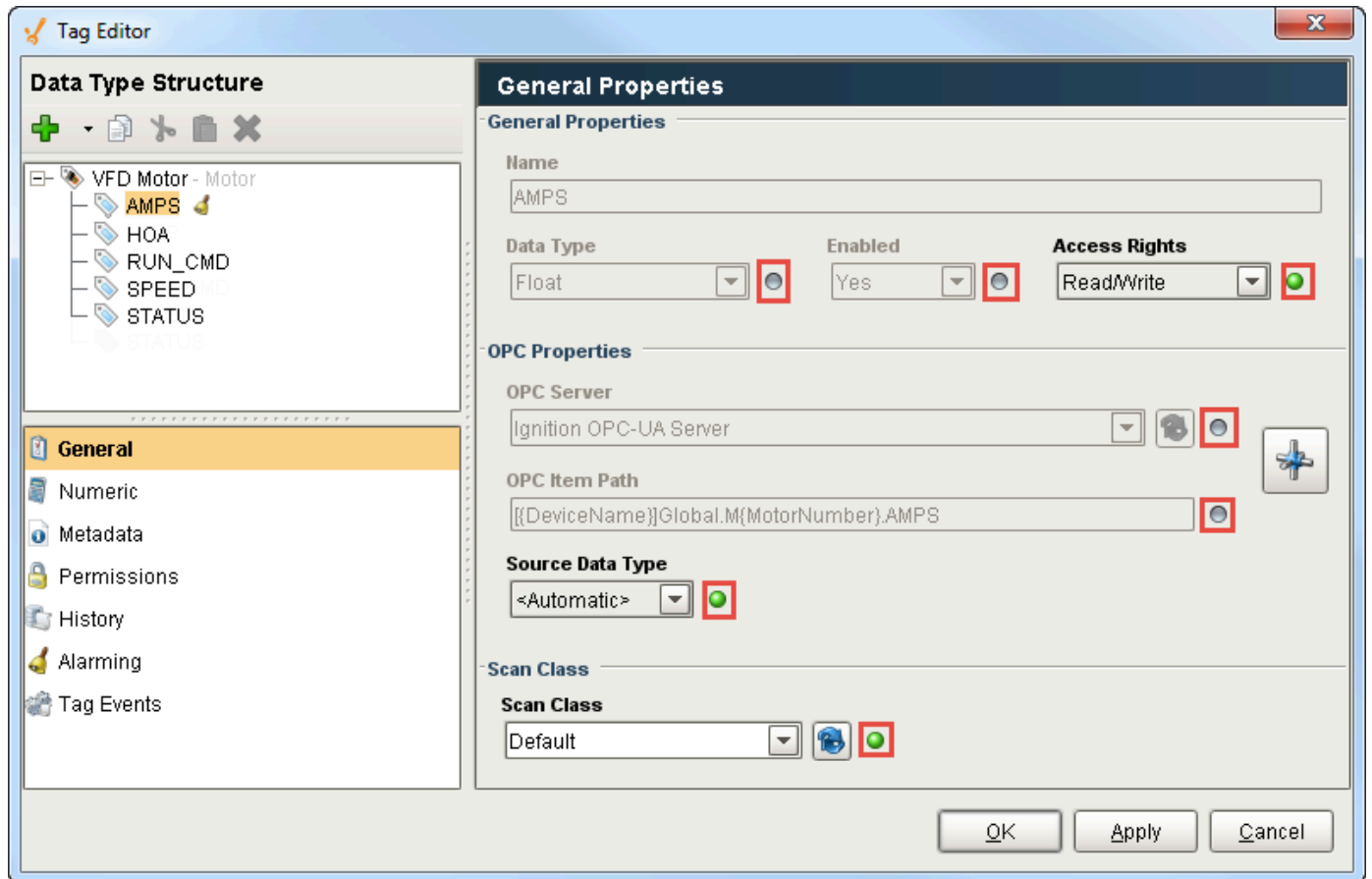


7. Now, you'll be able to see all the values for VFD1, and that the SPEED Tag was added.

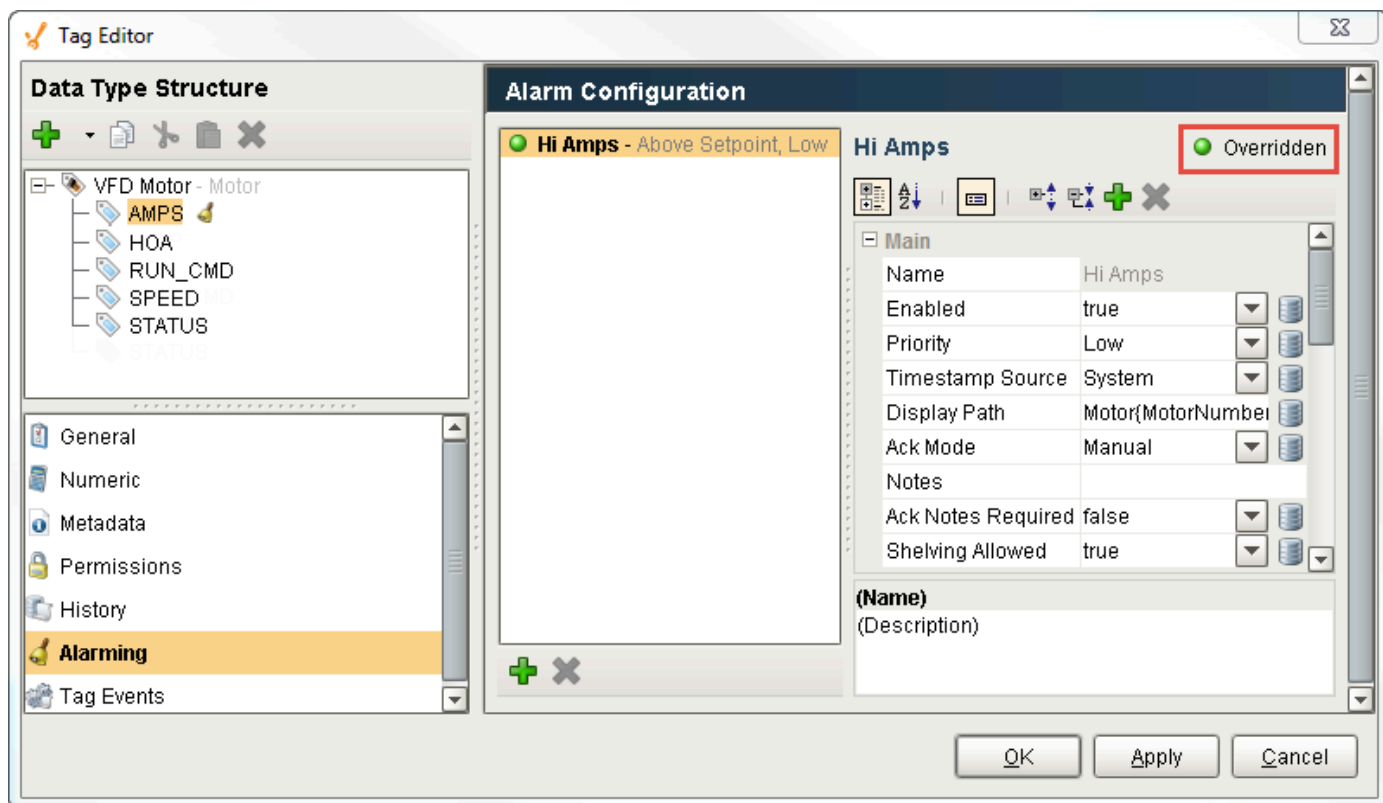


Another nice benefit of the inheritance feature is it allows you to modify or override some of the parameters of the parent. For example, since the VFD Motor has Motor as the parent, you can go to any of the Tags and override any of the settings of that data type.

1. Go to your **Tag Browser**, and double click on your **VFD Motor**. The Tag Editor will open and you'll see all your Tags.
2. Select any of your Tags (i.e., **AMPS**). You can **override any of the properties** by clicking the circle to the right of the property. When you click the circle, it changes to green allowing you to override the property inherited from the parent.



3. You can also turn on Alarming or History that wasn't initially turned on in the parent by simply using the override feature. Click on **Alarming**, and under Alarm Configuration, select your alarm and click the **Overwritten** circle at the top right of the window changing it from gray to green.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

UDT Composition



INDUCTIVE
UNIVERSITY

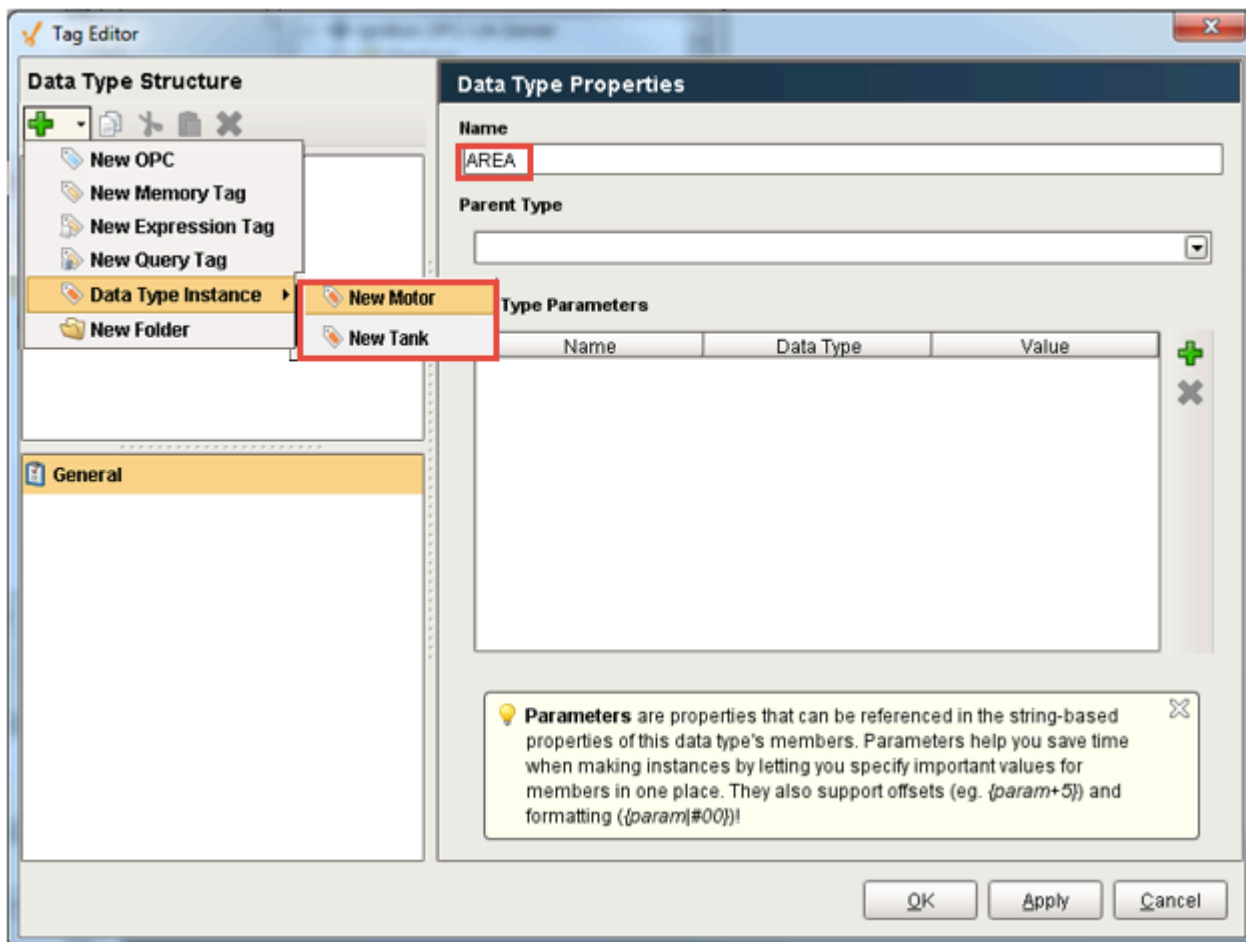
UDT Composition

[Watch the video](#)

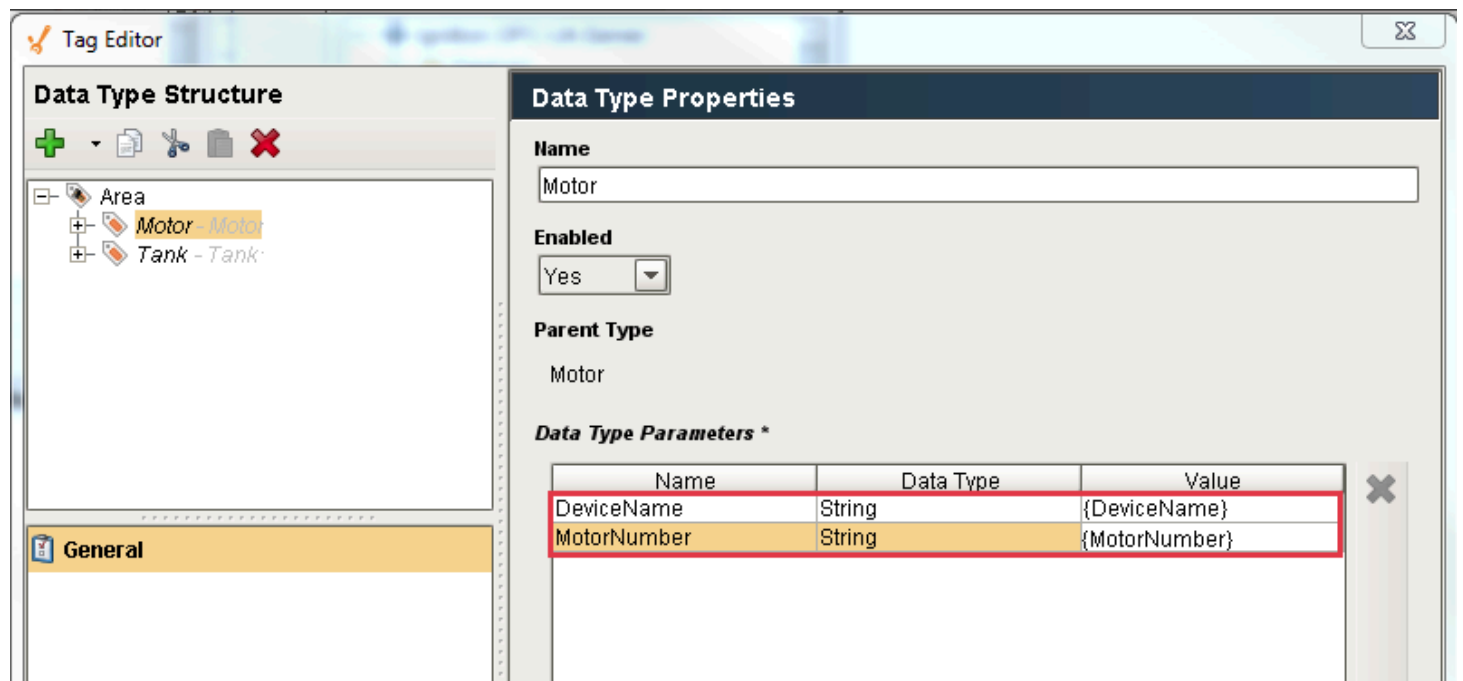
It's possible to set up UDT composition in Ignition where you are putting one UDT inside of another UDT.

In this example, let's use our a Motor and Tank data types that were already created and used in previous sections of this manual. We are going to create a third UDT called Area that will contain the Motor and Tank data types inside of it.

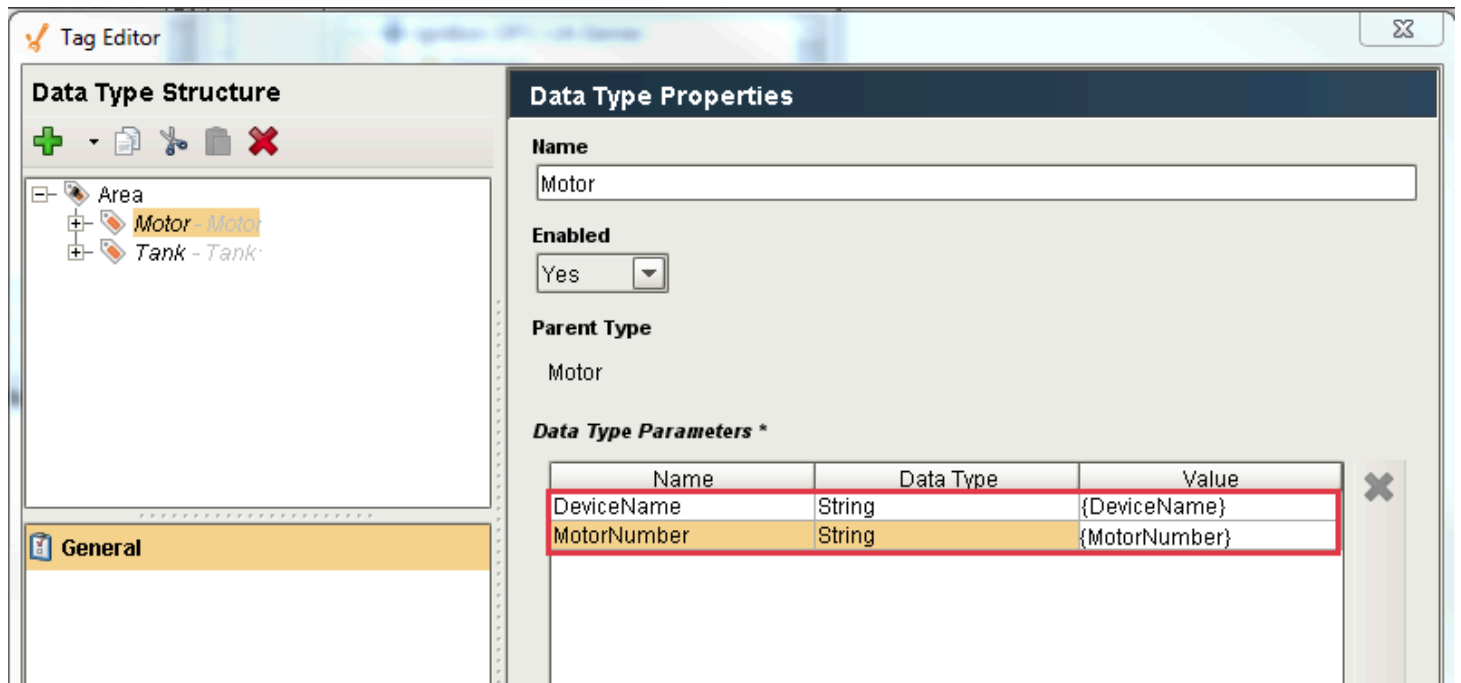
1. In the **Tag Browser**, right click on **Data Types** and select **New Data Type**. The Tag Editor window will open.
Assign the new data type a Name called **Area**.
2. Inside of the Area data type, create two data type instances, one for Motor and the other for Tank.
 - Click on the **green + icon**, select **Data Type Instance > New Motor**.
 - Click on the **green + icon**, select **Data Type Instance > New Tank**.
 - Rename both new data types to Motor and Tank respectively.



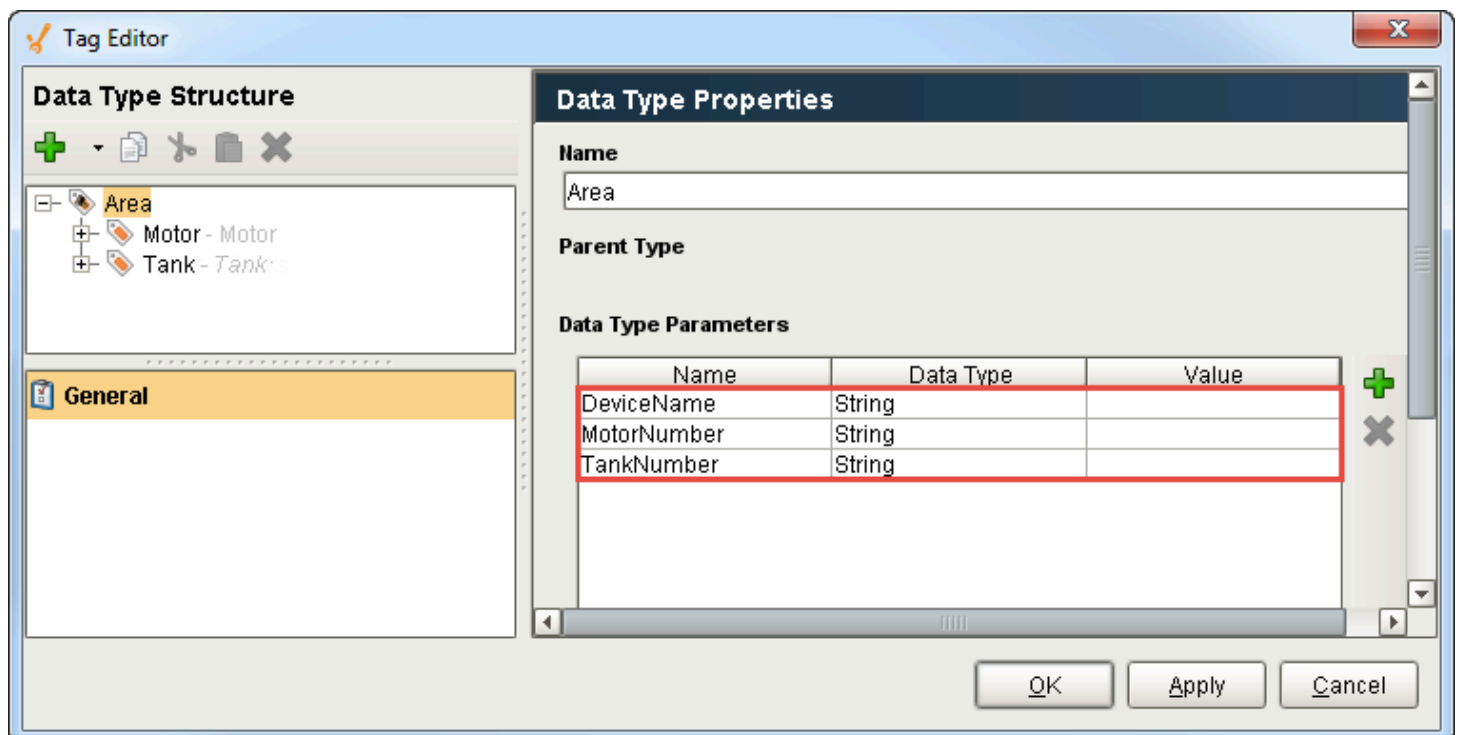
3. Now, you have both the Motor and Tank UDTs in the Area UDT. Motor has two parameters: **DeviceName** and **MotorNumber**.



Tank also has two parameters: **DeviceName** and **TankNumber**.



4. Every UDT you add inside another UDT, those instances have parameters that need to be specified and passed in from the parent UDT. In the Area UDT, create the **DeviceName**, **MotorNumber** and **TankNumber** parameters in the the Area UDT. (Even though both data types use the same **DeviceName** parameter, you only need to enter it once in the Area UDT).



5. The values in the Area UDT need to pass to the Motor and Tank data types so references need to be added.

Click on **Motor**, and enter the following references. Note that you do **not** have to type, but can edit the Value field, and press the Down Arrow on the keyboard:

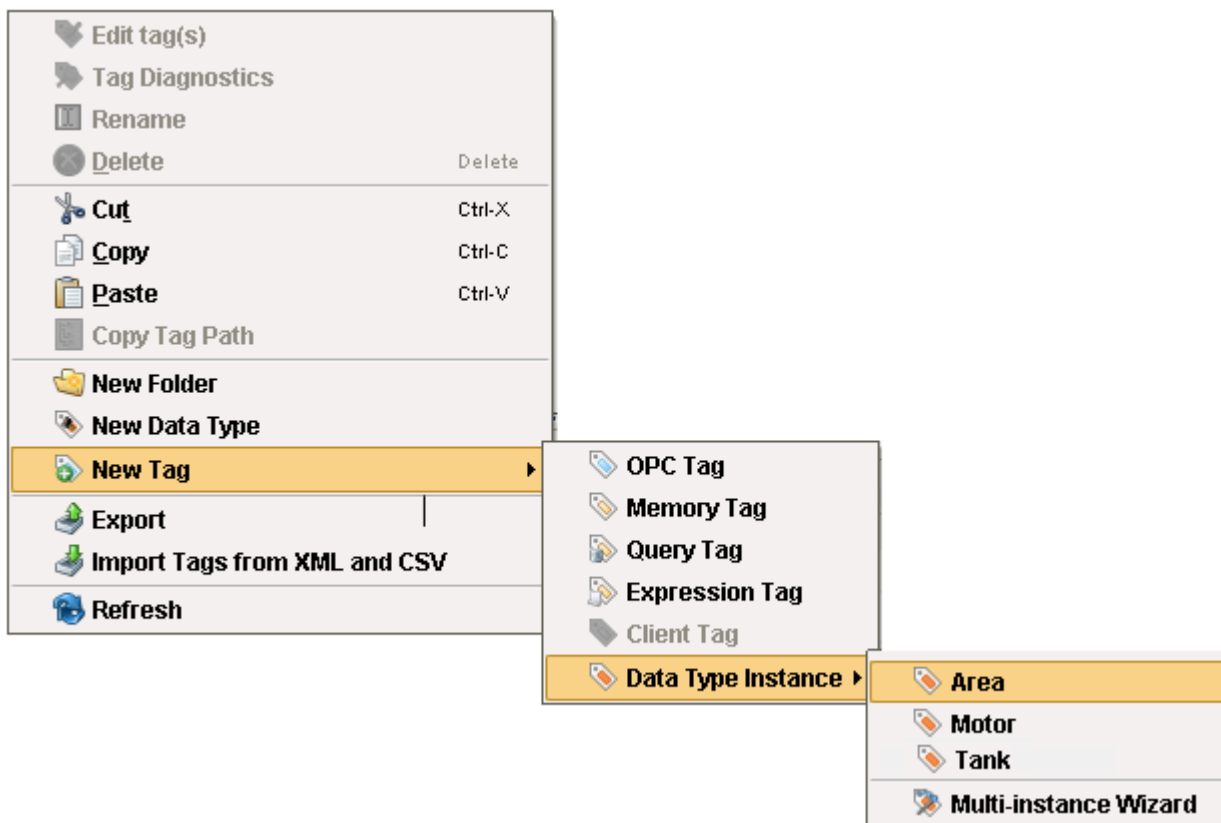
- For **DeviceName**, enter the value `{DeviceName}`.
- For **MotorNumber**, enter the value `{MotorNumber}`.

Click on Tank, and enter the following references:

- For **DeviceName**, enter the value `{DeviceName}`.
- For **TankNumber**, enter the value `{TankNumber}`.

Click **OK**.

- Now, that you added all your parameters, create a new instance of Area. In the **Tag Browser**, right click on **Tags**, select **New Tag > New Data Type Instance > Area**

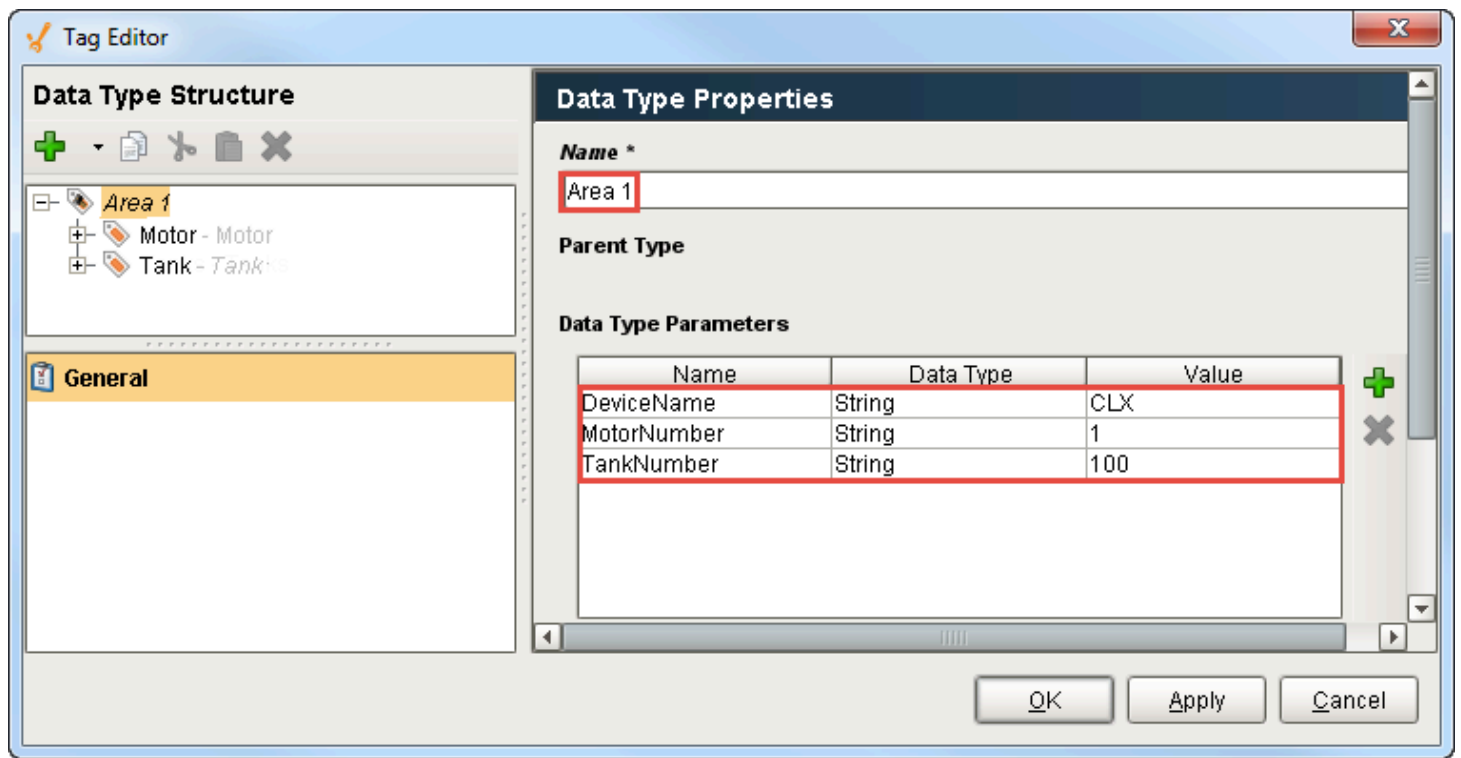


- The Tag Editor window will open. Enter a **Name** for the new instance name, called **Area 1**.

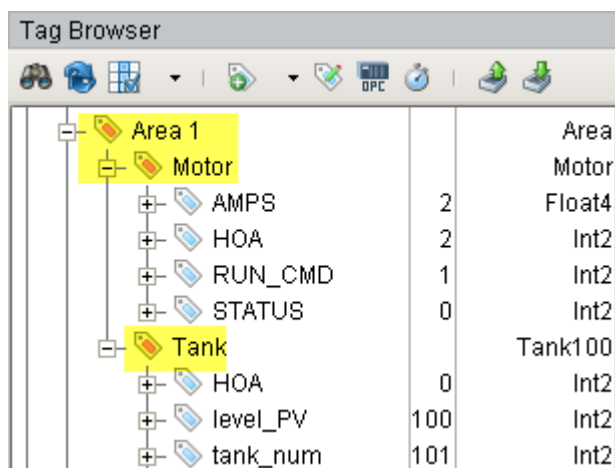
Enter the values for each parameter:

- For **DeviceName**, enter the value **CLX**.
- For **MotorNumber**, enter the value **1**.
- For **TankNumber**, enter the value **100**.

Click **OK**.



8. From the **Tag Browser**, you can see **Area 1** was created, and if you expand Area 1, it will show the two UDTs (Motor and Tank) and that values are coming through pointing to the appropriate places in the PLC.



9. Check the **OPC Item Path** in the Tag Tree to verify that Motor is pointing to **Motor 1** and Tank is pointing to **Tank 100**.

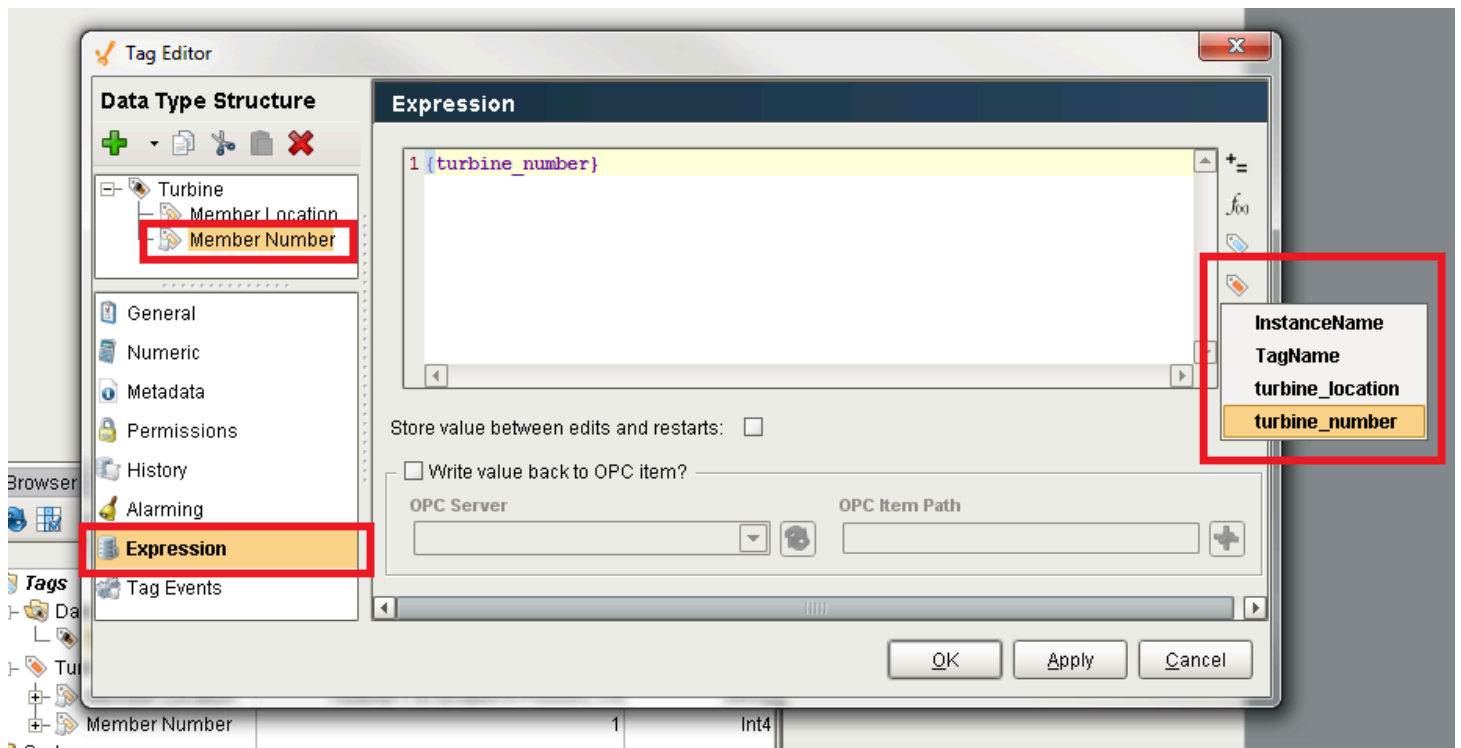
Tag Browser				
☒ Value ☒ Data Type ☐ OPC Server ☒ OPC Item Path ☐ Quality ☐ Last Change ☐ Scan Class ☐ Documentation ☐ Deadband ☐ Access Rights				
Area				n/a
Motor				n/a
Float4	2		[CLX]Global.M1.AMPS	
Int2	2		[CLX]Global.M1.HOA	
Int2	1		[CLX]Global.M1.RUN_CMD	
Int2	0		[CLX]Global.M1.STATUS	
Tank100				n/a
Int2	0		[CLX]Global.Tank100.HOA	
Int2	100		[CLX]Global.Tank100.level_PV	
Int2	101		[CLX]Global.Tank100.tank_n...	
Int2	0		[CLX]Global.Tank100.tank_st...	
Float4	80		[CLX]Global.Tank100.temp_d...	
Float4	65		[CLX]Global.Tank100.temp_h...	
Float4	81		[CLX]Global.Tank100.temp_PV	
Float4	25		[CLX]Global.Tank100.temp_SP	
Int2	0		[CLX]Global.Tank100.valve_c...	
Int2	0		[CLX]Global.Tank100.valve_s...	

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Data Type Parameters in Expressions

It is possible to use the value of data type parameters directly in expression bindings within a UDT. Parameter references can be quickly inserted into an expression by clicking on the UDT Parameters button on the right of the expression area.



Syntax

The syntax for data type parameter expressions differs from Tag and property references when using string values. **Quotation marks must be used when referencing string parameters.** The reason for this is that data type parameter references are replaced before any expressions are evaluated. This differs from Tag and property references seen elsewhere in Ignition.

If the data type of the parameter is not a string, then quotation marks are not required, but can be used.

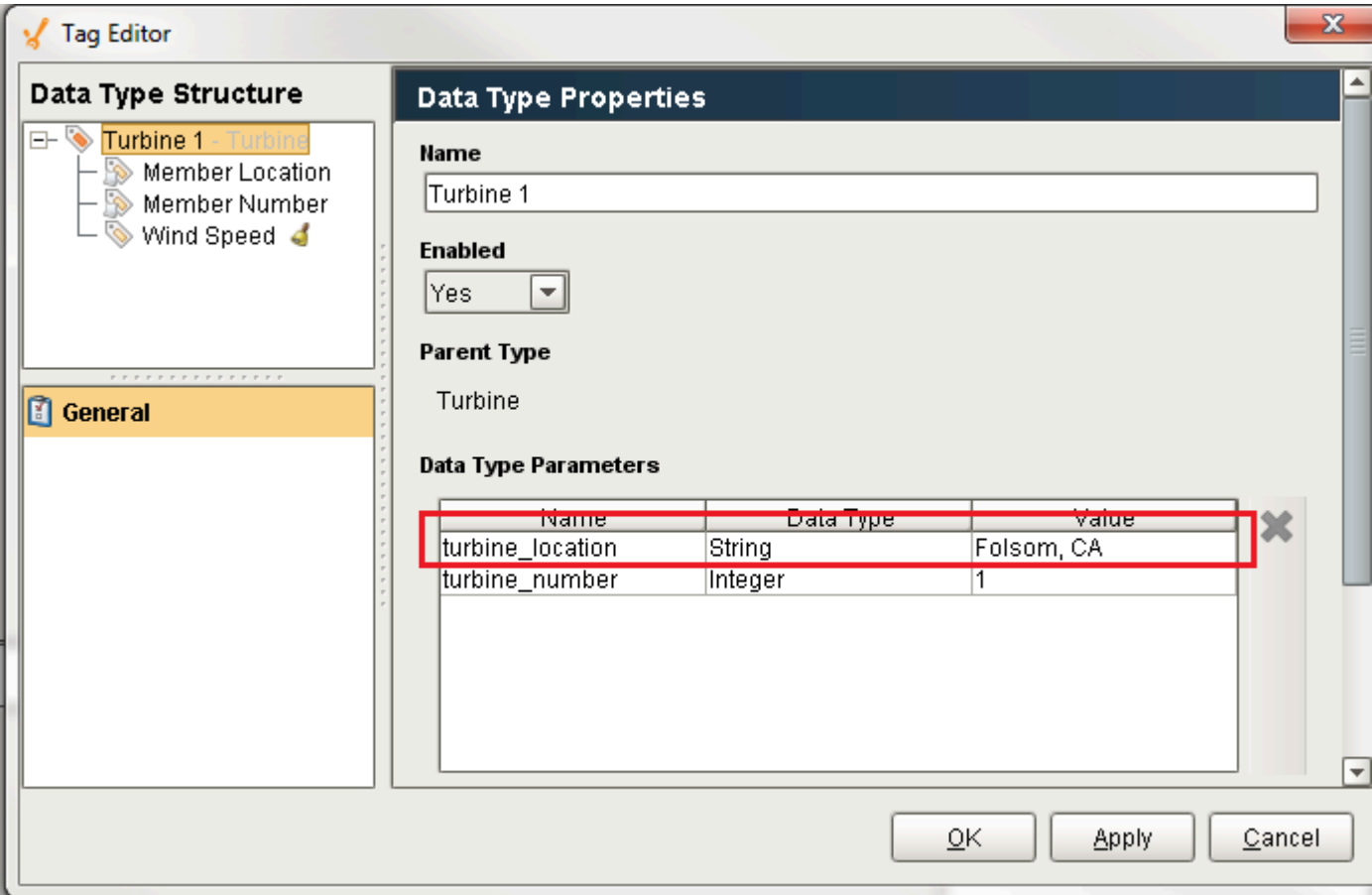
It does not matter if double quotation marks or single quotation marks (" versus ') are used as long as a matching closing quotation mark is present.

Syntax Code Examples

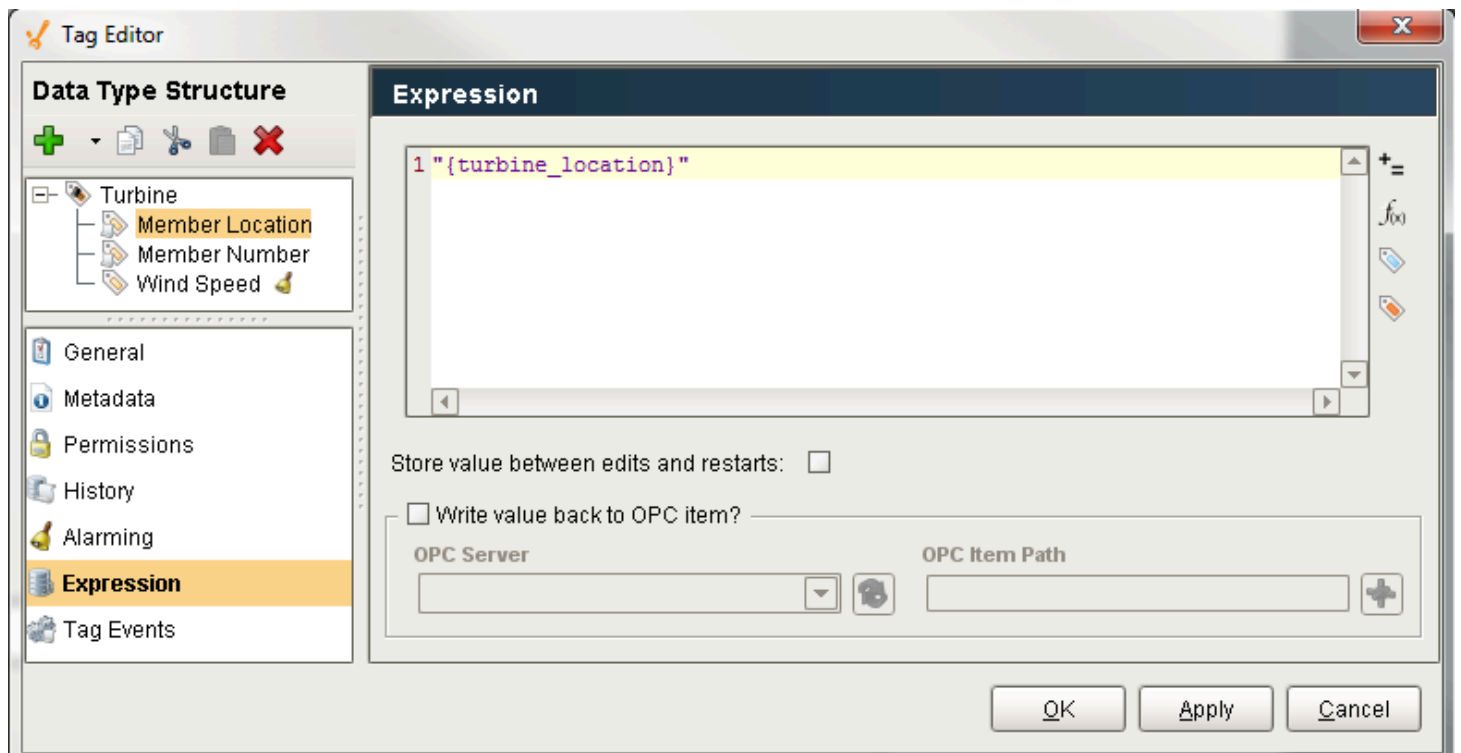
```
"{string parameter}" //This expression will evaluate successfully
'{string parameter}' //This expression will evaluate successfully as well
```

Examples

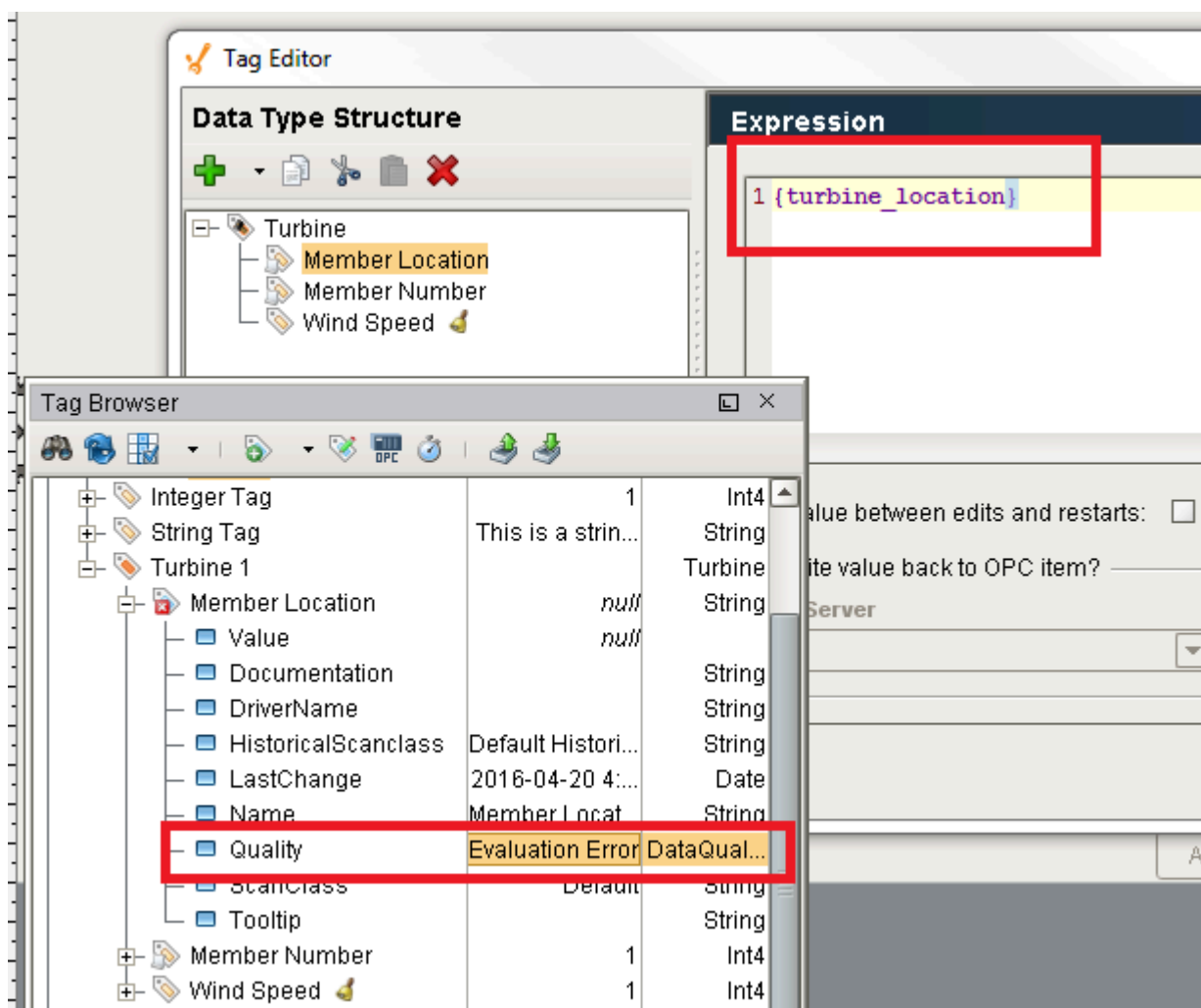
Below is an instance of a Turbine UDT. It contains a string parameter named "turbine_location".



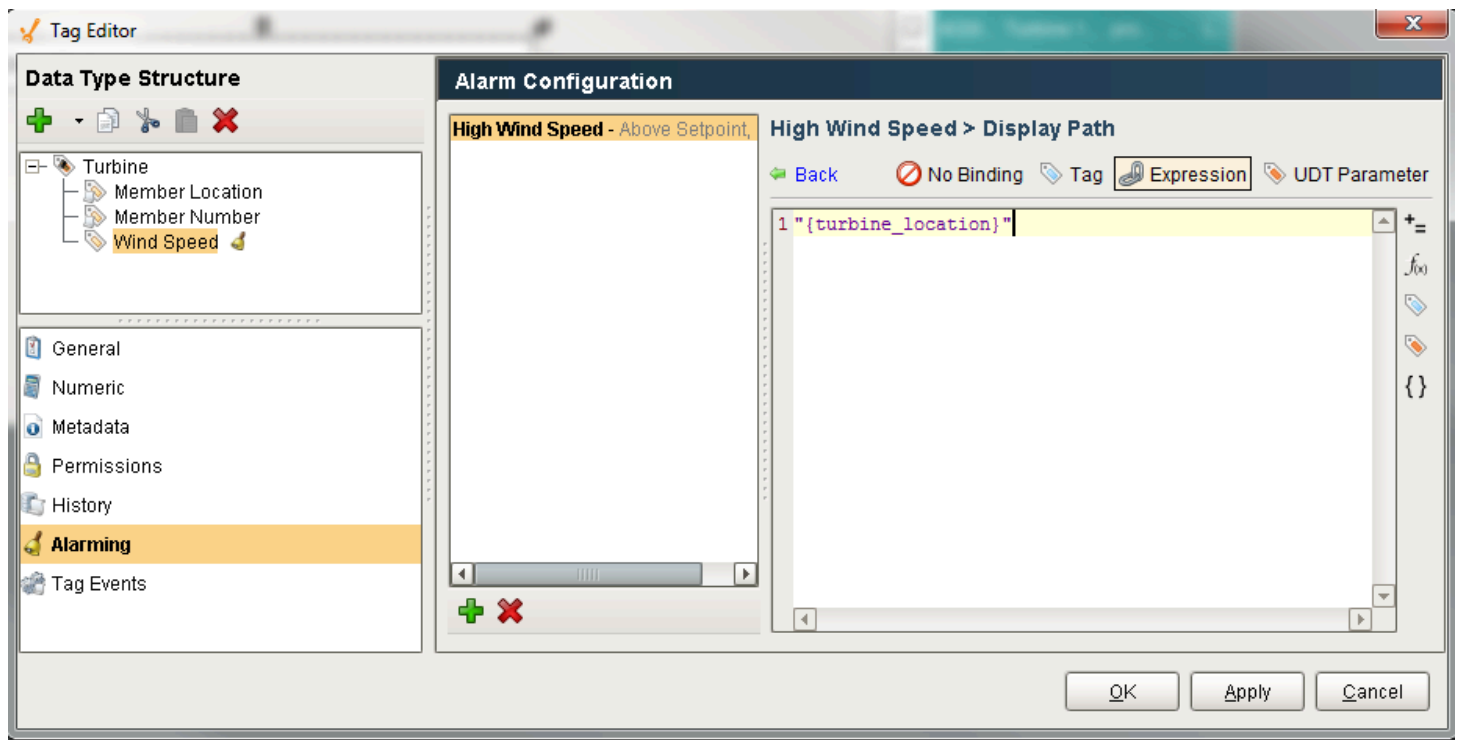
Inside the Turbine UDT is an expression Tag named Member Location. To show the value of the **turbine_location** parameter on Member Location, the following expression would be used on the UDT definition.



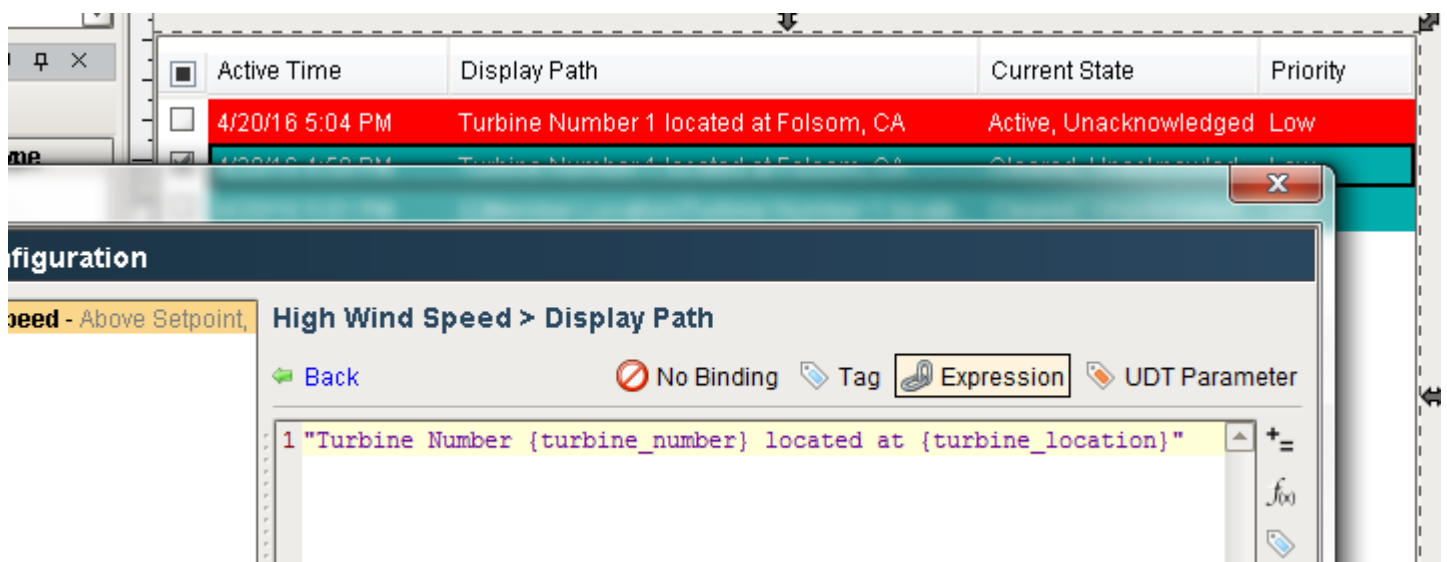
Omitting the quotation marks would result in an evaluation error.



The same syntax should be applied to parameters in bindings on alarm properties. Below is an example using the same UDT, but the expression is instead located on the Display Path property of an alarm.

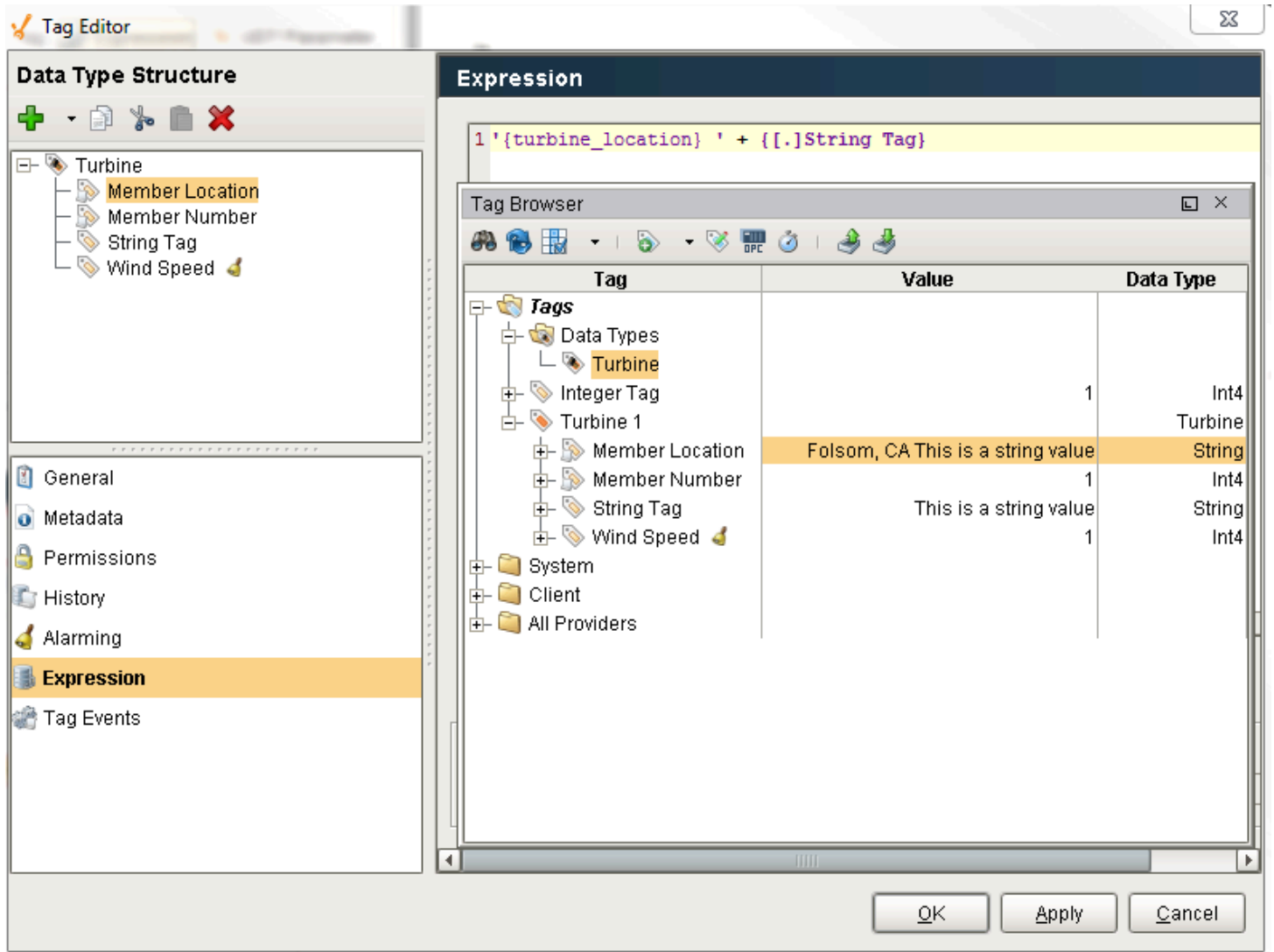


Additionally, non-string parameters and static values can be included in the same quotation marks.



Combining Parameters and Tag References

Because parameter and Tag references differ in syntax, some consideration must be made when attempting to use both in the same expression. Tag references must not be placed inside of quotes. After adding a string Tag to the Turbine UDT, a reference to the Tag can be added to Member Location's expression.



Adding quotation marks around the Tag reference would result in the literal value of the reference appearing in the value of the Tag.

Data Type Structure



- Turbine
 - Member Location
 - Member Number
 - String Tag
 - Wind Speed

- General
- Metadata
- Permissions
- History
- Alarming
- Expression**
- Tag Events

Expression

```
1 '{turbine_location} {[.]String Tag}'
```

Tag Browser



Tag	Value	Data Type
Tags		
Data Types		
Turbine		
Integer Tag	1	Int4
Turbine 1		Turbine
Member Location	Folsom, CA {[.]String Tag}	String
Member Number	1	Int4
String Tag	This is a string value	String
Wind Speed	1	Int4
System		
Client		
All Providers		

OK

Apply

Cancel

Quick Reference

Reference Type	Require Quotes?	Expression Example
Static String	Requires Quotes	<code>"This is a string"</code>
String UDT Parameter	Requires Quotes	<code>"{turbine_location}"</code>
String Tag Reference	No Quotes	<code>{[.]String Tag}</code>
String UDT Parameter with Static String	Requires Quotes	<code>"The turbine is located at {turbine_location}"</code>
String UDT Parameter with String Tag Reference	Partial Quotes	<code>"{turbine_location} " + {[.]String Tag}</code>
String Parameter, Static String, and String Tag Reference	Partial Quotes	<code>"The Turbine " + {[.]String Tag} + " {turbine_locatio</code>

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

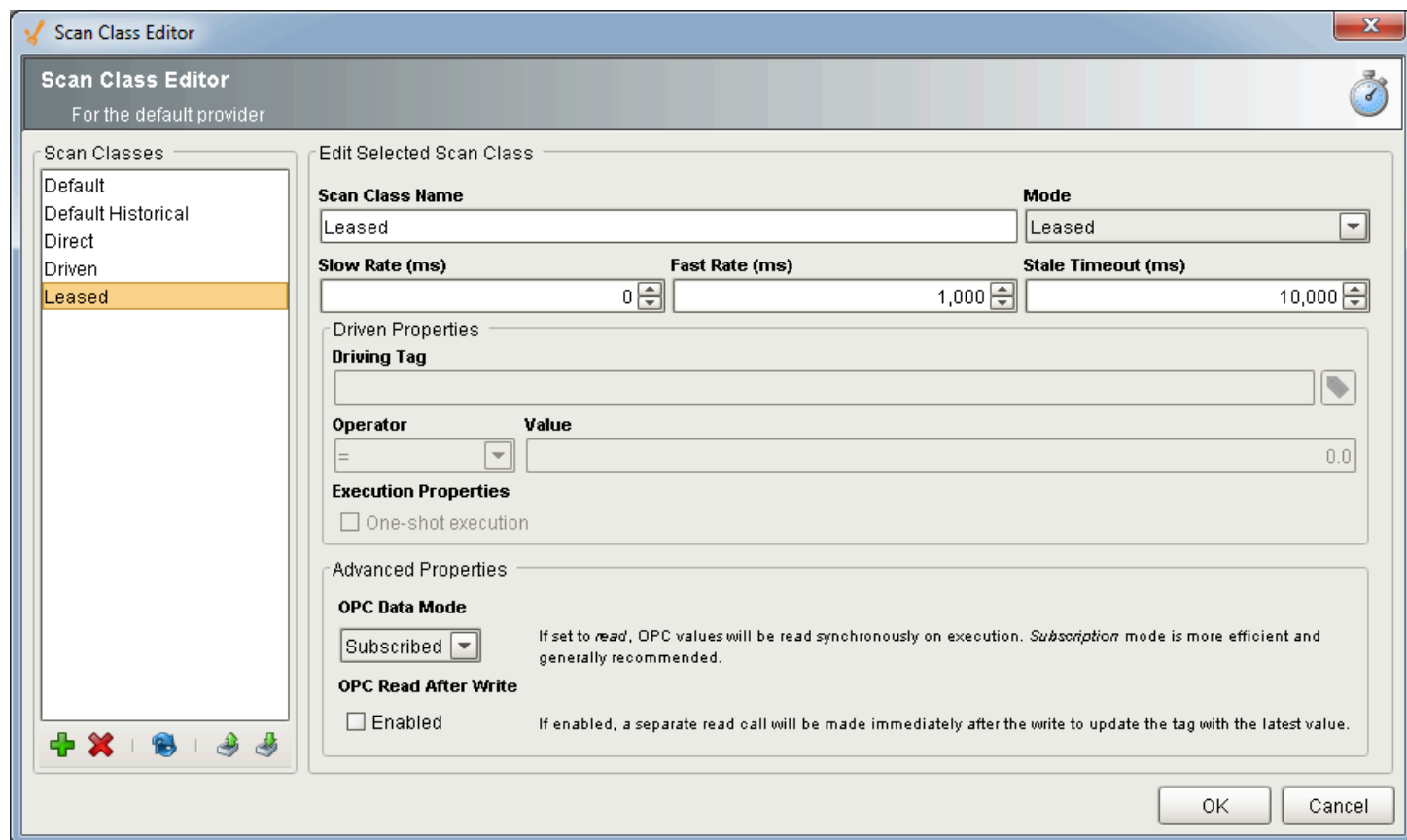
For documentation on the latest stable release, see the [latest version](#) (8.3).

Scan Classes

What is a Scan Class?

Scan classes dictate the rate of execution of Tags, and therefore play a crucial role in the design of large and high-performance systems. It is beneficial to have multiple scan classes so you can set groups of Tags to run at different rates. Since it takes up computer resources to constantly fetch new Tag values, this frees up a lot of resources if many of the Tags are running slower. Some Tags you may wish to see updated at 250-500ms, while others may not be so crucial and may only need an update every 10-30 seconds.

Creating different scan classes allows you to organize your Tags into groups that subscribe (continually poll) at different rates. It is good practice to put some forethought and planning into the organization of your Tags and scan classes.



Tag Execution by Scan Class

Tags are executed by scan classes inside of a Tag Provider. In a typical system, there are a number of scan classes and one or more Tag Providers. Tag Providers can be either internal or external.

- **Internal provider** stores the Tag configuration and values inside Ignition. Tags inside Internal Providers are available to other Ignition Gateways that use the **Remote** provider type.
- **External provider** stores Tag configuration and values in a database. Tags stored in an external provider are available to all Ignition installations that have access to that database. One of the installations must be specified as the Tag's driver. The driving system has a copy of the scan class that it executes, which in turn evaluates the Tag. The value is stored to the database and all of the other installations are notified of the new value.

! THE WORDS SCAN CLASS "TYPES" AND "MODES" USED INTERCHANGEABLY

When referring to scan classes, we often use the words scan class "type" and "mode" interchangeably throughout this manual. Ignition has three scan class modes (i.e., Direct, Driven, and Leased), but there can be different variations of these modes that can change their behavior. For example, the Driven scan class has "one-shot" and "any change" options that you can set, and we often refer to these as different "types" of scan classes as well. When you create a new scan class, you will select one of the three "modes" and any properties you need to build your scan class.

Types of Scan Classes



INDUCTIVE
UNIVERSITY

Types of Scan Classes

[Watch the video](#)

Tags are assigned a scan class that determine how often they update. For example, how often an OPC value is polled from the PLC, how often an Expression Tag calculates its expression, and how often a Query Tag runs its query. It's easy to create scan classes in Ignition for just about any scenario you can think of. Scan classes are extremely powerful and flexible, and you can create them based entirely on your individual business requirements.

There are three different scan class modes in Ignition that you can use to build your own, each with a different options:

- **Direct** - executes at a fixed rate. Every Tag that uses a direct scan class will poll the PLC at the slow rate setting at all times (24x7x365). Ignition provides two Direct scan classes to help you get started, but you can change them however you'd like. It is recommended to adjust the "Default" scan class to a slower rate as all new tags are automatically added to this scan class.
 - **Default** - runs at a rate of 1 second.
 - **Default Historical** - runs at a rate of 10 seconds.
- **Driven** - executes fast or slow based on a condition. The rate of the Driven scan class is determined by a condition based on the value of the selected **Driving Tag**. This condition is a simple comparison between a driving Tag's value and a static number (equal to, greater than, etc.). While the condition is true, the scan class executes at the fast rate. If false, it runs at the slow rate. (It's useful to keep in mind that the driving Tag can also be an Expression Tag that performs complex calculations and references other Tags. In this way, it's possible to create robust scan class triggering).
 - **Driven One-shot** - If you want to read Tag values only once on certain conditions (instead of polling), there are two properties that change the behavior of the Driven mode: the **One-shot** property, and the **Any Change** Operator option. Using either of these, the scan class does not run at either the fast or slow rate. Instead, it will be triggered once each time your condition is met. **Any Change** will execute each time the driving Tag value changes, and **one-shot** will execute once when the comparison condition is true and not again until the condition becomes false and subsequently true.
- **Leased** - executes as on-demand polling. It works similar to the Driven class except there is no driving Tag. Instead, the driving mechanism is whether a Tag is being used on an open window. That is, if a user is looking at a Tag, it runs at the fast rate, and if the Tag is not being shown, it is moved to the slow rate.

CAUTION

When using scan classes with both the fast and slow rates (Driven or Leased), a **0** poll rate can be specified. This means the Tags in this scan class will not poll at all. Do not set a **0** poll rate on Tags that are storing history or need other regular updates.

Scan classes are created and edited in the Designer, and described in detail in [Editing Scan Classes](#).

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

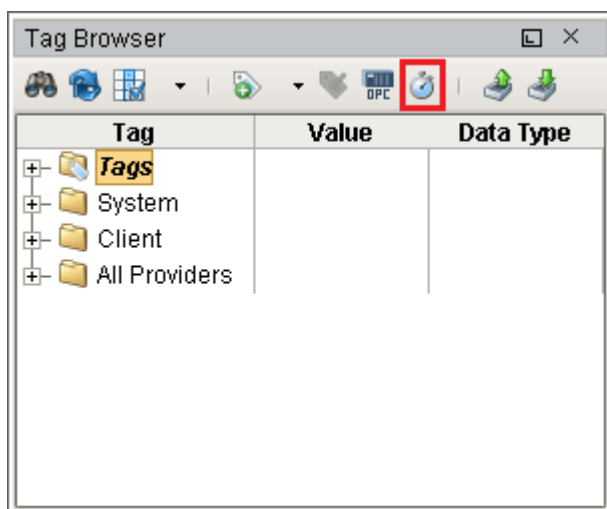
Editing Scan Classes

Adding and Editing Scan Classes

Adding and editing scan classes is really easy in the Designer once you understand how the different **scan class modes** work. It's just a matter of choosing which scan class mode you want to use for your Tag, and entering the properties for your scan class.

To Add and Edit a Scan Class

1. In the **Tag Browser**, click on the **Timer** icon () to open the **Scan Class Editor** window.



2. A list of already configured scan classes appear on the left side of the window, and configuration settings on the right. Click on a scan class to edit it, and to add a scan class, simply click the **green +** icon on the lower-left of the window.
3. For creating a new scan class, specify the **Name** of the scan class, and select a **Mode**: **Direct**, **Driven**, or **Leased**.
4. Enter the **Slow**, **Fast**, and **Stale Timeout** rates, as applicable. (Not all the rates are editable based on the type of scan class you are using).
5. If you're using a Driven scan class, enter the driving properties if they are applicable to the type of scan class you're using:

- **Driving Tag** - select the driving Tag from the dropdown list on the right side.
- **Operator** - determines how the Value property should be compared to the Driving Tag's value.
- **Value** - uses the Operator property to determine if the scan class should execute at the slow or fast rate.
- **Execution Properties** - mark the checkbox if this is a one-shot execution.

6. Enter Advanced Properties (optional)

- **OPC Data Mode** - dictates how the OPC values are obtained, by Subscription or Read.
- **OPC Read After Write** - mark the checkbox to enable read after write.

For more information on how to set up each scan class mode refer to Direct, Driven, Leased, and Driven One-shot sections.

A screenshot of the Scan Class Editor is shown below. Each Scan Class Editor property is described in detail in the table below.

Scan Class Editor

Settings that edit a selected scan class.

Setting	Description
Scan Class Name	Unique name of the scan class.
Mode	• Direct The scan class executes at a fixed rate, defined by the slow rate setting. • Leased The scan class executes at the fast rate when any of the Tags it contains are subscribed and visible in a Client window. If no Tags are subscribed, the scan class runs at the slow rate. • Driven The rate of the scan class is based on the value of a driving Tag. The condition is a simple comparison between a Tag value and a number. If the condition is true, the scan class will execute at the fast rate. If false, it will run at the slow rate. There are two exceptions to this: The Any Change operator, and One-shot mode. Using either of these, the scan class will not run at a rate. Instead, it will be triggered by a change in the driving Tag's value. Keep in mind that the driving Tag can be an Expression Tag that performs complex calculations and references other Tags. In this way, it's possible to create robust scan class triggering.
Slow Rate	Base update rate, specified in milliseconds, at which Tags will be executed. Note: If the rate is set to 0, the scan class will not be executed. It is common for leased and driven modes to use 0 as a slow rate in order to achieve an on/off effect.
Fast Rate	Used by the Driven and Leased modes, this is the faster rate that the Tags will be executed at when those modes are active. Note: If the rate is set to 0, the scan class will not be executed.
Stale Timeout	How long to wait before the Tags in the scan class are determined to be stale (not running). This is calculated off of the last expected execution time of the scan class, and is particularly important for scan classes executed by

Setting	Description
	other drivers through the external Tags provider. This property is not used by internal providers.

Driven Properties

Used by the driven mode to determine when the scan class should run at the fast rate.

Setting	Description
Driving Tag	Which Tag will determine when the rate of the scan class should be changed. The Driving Tag should not use this same scan class. The Driving Tag should be on a separate scan class that runs at a faster rate than the Driven scan class. This means that changes to the Driving Tag will be recognized by the system faster, and the change in rate on the Driven scan class will occur faster.
Operator	How the Value property should be compared to the Driving Tag's value. If the comparison is true, then the Fast Rate will be used by the Scan Class ,otherwise, the Slow Rate will be used. The Any Change operator works differently than the other operators: The scan class will execute immediately whenever the driving Tag changes value. Using the Any Change operator means that the scan class no longer uses the Slow Rate or Fast Rate properties.
Value	Used by the Operator property to determine if the Scan Class should execute at the slow or fast rate.
One-shot execution	One-shot will execute once when the comparison condition is true, and not again until the condition become false, and subsequently true.

Advanced Properties

Settings that subtly affect how the scan class operates.

Setting	Description
OPC Data Mode	This mode dictates how OPC values are obtained. The default mode, Subscription, is preferred because it is much more efficient than a read. • Subscribed All OPC Tags in the scan class will be subscribed according to the scan class rate. Values will come in asynchronously as they change. • Read Tags will not be subscribed, but will instead be synchronously read each time the scan class executes. This operation is less efficient, but allows more precise control over when values are obtained. This mode is particularly useful when collecting data over a slow or expensive connection for display. When combined with the one-shot execution mode above, and a static Tag tied to a momentary button, it's easy to create a manual refresh button on a screen that pulls data on-demand.
OPC Read After Write	When enabled, a read request will be sent immediately after a write request. This means that the value on the Tag will be updated much quicker to reflect the latest written value. Enabling this property is less efficient as a single write to a Tag becomes two separate requests. This is especially helpful with slower scan classes as the Tags will show the latest value quicker than the normal execution would allow.
OPC Optimistic Writes	New in 7.9.2 If enabled, written values will be applied to the OPC Tag immediately. Normally, the system must receive confirmation that a write was successful from the device before the OPC Tag's value would change. This property changes the behavior by assuming the write went through unless the next read value or subscription update proves otherwise. Enabling this will make writes appear to execute much quicker. Works in conjunction with the OPC Optimistic Write Timeout property below. If the OPC Tag does not receive confirmation that the new write was successful within the timeout, the Tag will fallback to the last known value. While in an ambiguous state, the Tag will have a quality of "Good

Setting	Description
	(Provisional)". This setting can be paired with the OPC Read After Write: the Tag will assume the newly written value, while an asynchronous read request is quickly sent out to confirm the write went through. While the write is pending, values received from subscription activities will override the fallback value. Assuming an initial value of 0, if a write of 10 is applied to the OPC Tag, then the Tag will show a value of 10 until the system can confirm the new value. If a subscription update then returns a value of 5, the OPC Tag will fallback to 5.
OPC Optimistic Write Timeout	New in 7.9.2 The timeout period for Optimistic Writes. A value of 0 effectively disables the fallback functionality: the new value is maintained on the Tag until the next read or subscription activity.

Historical Scan Classes

Historical scan classes are simply standard scan classes used by Tags to store history. By using separate scan classes for status and history, it's possible to maintain a Tag's status at a fast rate, without storing large amounts of history unnecessarily.

Despite the fact that there is not a technical differentiation between standard and historical scan classes, it is recommended that you create separate scan classes for each purpose and name them in a manner that indicates their usage. It is common to modify scan classes in order to affect a large number of Tags, and without a consistent distinction it may be possible to affect Tag execution in unexpected ways.

OPC-UA Stale Threshold



INDUCTIVE
UNIVERSITY

OPC-UA Stale Threshold

[Watch the video](#)

Ignition uses the OPC-UA Stale Threshold setting to determine when updates from a driver have taken too long to complete. This time period will be calculated as the fastest sampling rate for that node multiplied by the Stale Threshold setting value. It's this multiplier on the scan class that determines when the Tag can be marked as stale or not. The default for the Stale Threshold value is set to 5. This is a very important setting because when the network is going slower or the PLC is taking a long time, those Tags need to be marked in Ignition as stale so that the operators know not to trust those values on the screen.

NOTE

The OPC-UA Stale Threshold is one setting is for the entire Ignition server.

To Edit the Stale Threshold

- 1. Go to the **Configure** section of the Gateway, and select **OPC-UA Server > Settings**.

The **Settings** page is displayed.

- 2. Scroll down to the **Other** section and then to the **Stale Threshold** setting.

This is a multiplier by which the server determines that updates from a driver have become stale.

Other	
Stale Threshold	5 The multiplier by which the server determines that updates from a driver have become stale. This period will be calculated as the fastest sampling rate for that node multiplied by this settings value. (default: 5)
Allow Untrusted Certificates	☒ Whether or not to automatically accept incoming certificates or to rely upon them being placed into the trusted certificate list before allowing a connection. (default: true)
Auto-cast Writes	☒ Attempt to auto-cast incoming writes to the correct DataType before rejecting them with a Bad_InvalidType StatusCode. (default: true)
Read Only When Inactive	☐ Operate in read-only mode when this node is the inactive node of a redundant pair. (default: false)

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Setting a Tag's Scan Class



INDUCTIVE
UNIVERSITY

Setting a Tag's Scan Class

[Watch the video](#)

Each Tag in Ignition is assigned a scan class which dictates the polling rate and conditions on which the Tag will be evaluated. For example, the scan class will dictate how often that value is going to poll from the PLC if it's an OPC Tag, or how often the expression is going to run if it is an Expression Tag, or how often the value is going to query the database if it is a Query Tag. Whatever type of Tag you're using, you can set a scan class on that Tag. You can also specify both a Realtime scan class and a Historical scan class for each Tag.

The three different **scan classes** that specify the rate of execution of Tags are as follows:

- **Direct** executes based on the slow polling rate setting at a fixed rate.
- **Driven** executes based on a condition.
- **Leased** executes according to on-demand polling.

To Set a Scan Class for a Tag

1. In the **Tag Browser**, right-click on any Tag, and click **Edit tag(s)**.

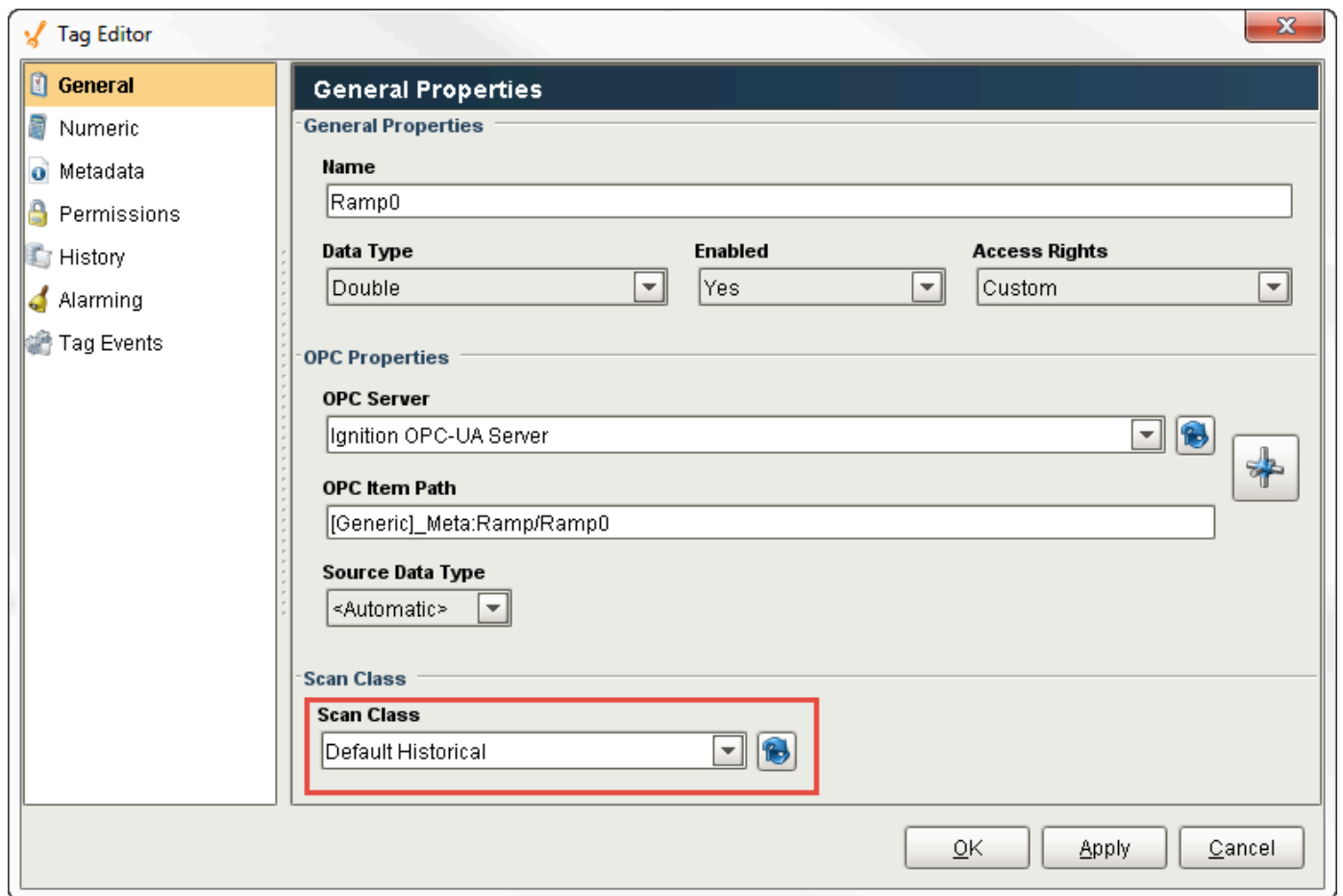
The **Tag Editor** window is displayed.

2. On the **General** tab, choose a scan class from the **Scan Class** drop-down for the selected Tag.

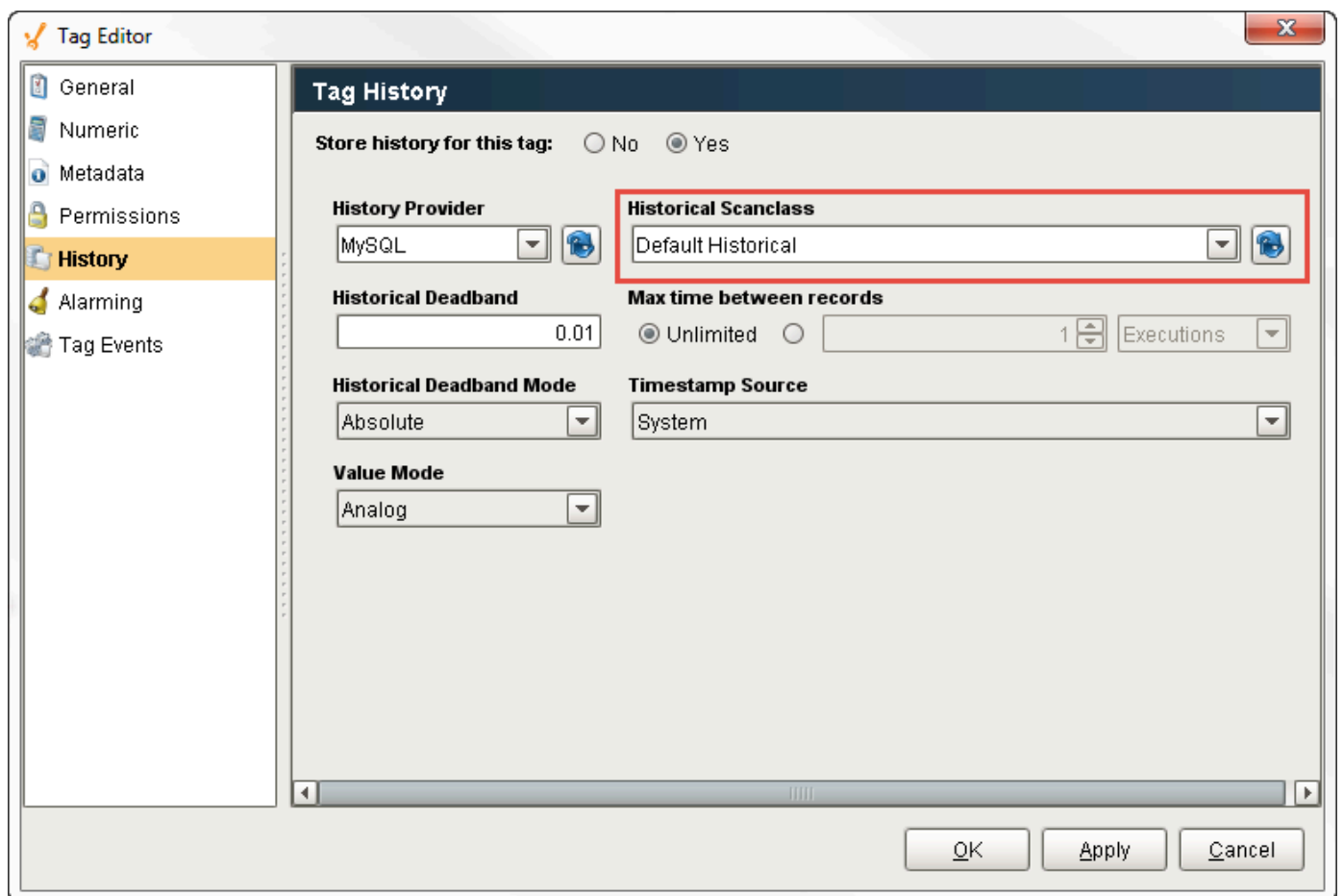
For example, choose the **Default Historical** scan class.

3. Click **OK** to save it.

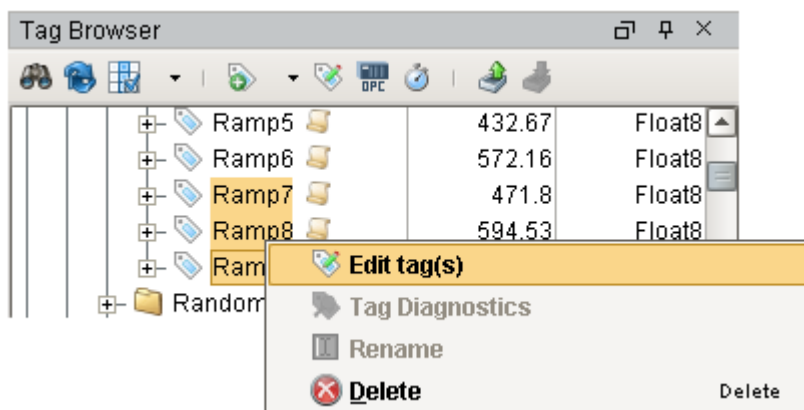
You will immediately see the Tag updating at a different rate.



4. The Tag also uses the scan class to determine how fast to log data for the Historian. The Historical Scan Class doesn't affect how fast the values get polled from the PLC, but affects how fast the data gets logged to a database. On the **History** tab, you can set the Historical Scan Class to **Default Historical**.



As you can see, setting scan classes on a Tag is extremely easy. You can even select multiple Tags in the Tag Browser by right clicking to edit the Tags. This opens the Tag Editor, and simply set the scan class for all the selected Tags at the same time.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Direct Scan Class



INDUCTIVE
UNIVERSITY

Direct Scan Class

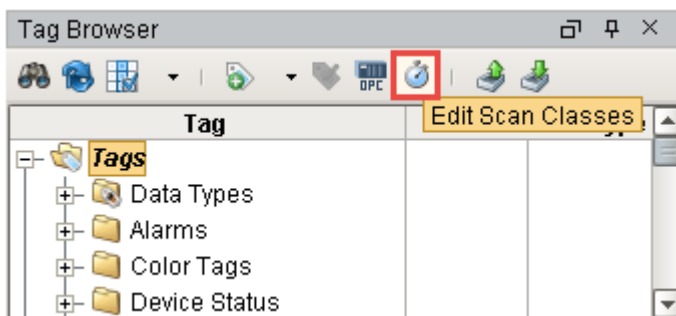
[Watch the video](#)

The Direct scan class executes at a fixed rate, which is defined by the slow polling rate setting. Every Tag that uses the Direct scan class will poll the PLC at the Slow Rate setting at all times (24x7x365).

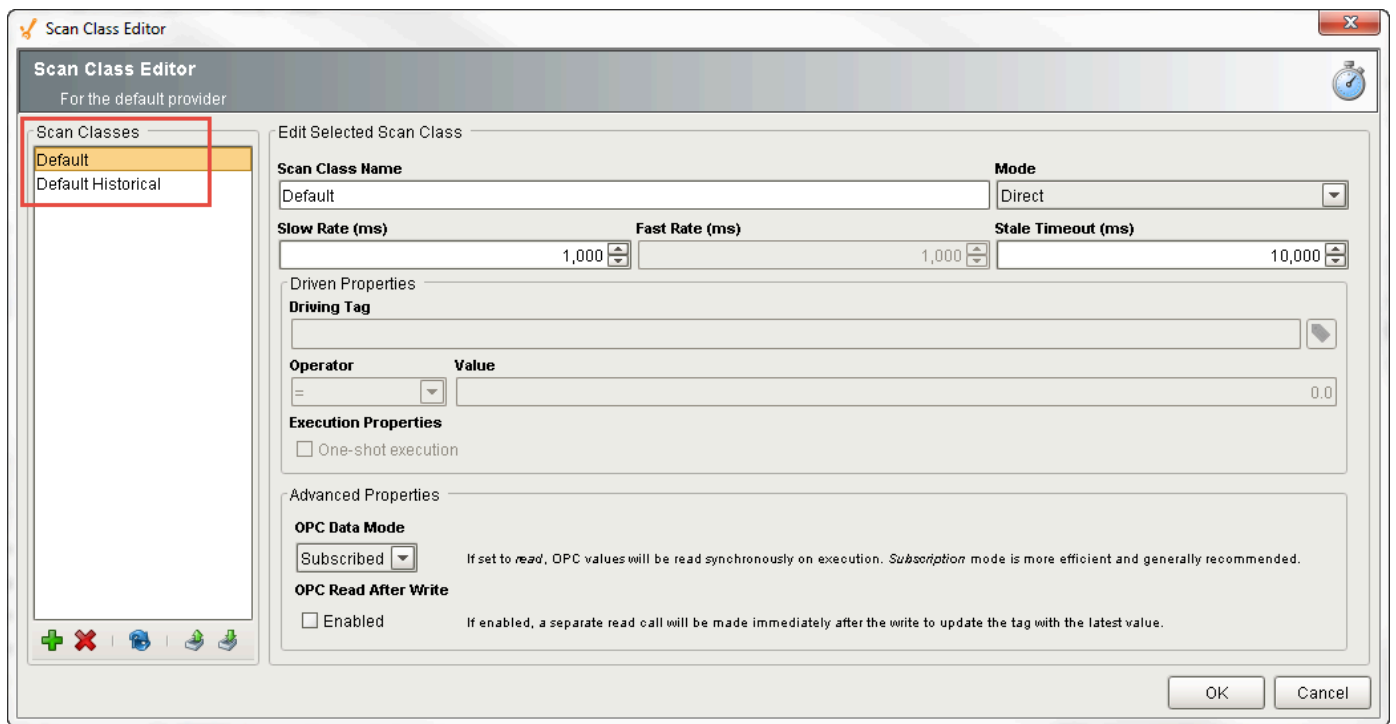
Ignition comes with two default Direct scan classes called **Default** and **Default Historical**. You can use any Tags in your system, including multiple Tags, for creating a Direct scan class.

To Add a Direct Scan Class

1. Open the **Designer**, in the **Tag Browser**, click on the **Timer** () icon to open the **Scan Class Editor**.



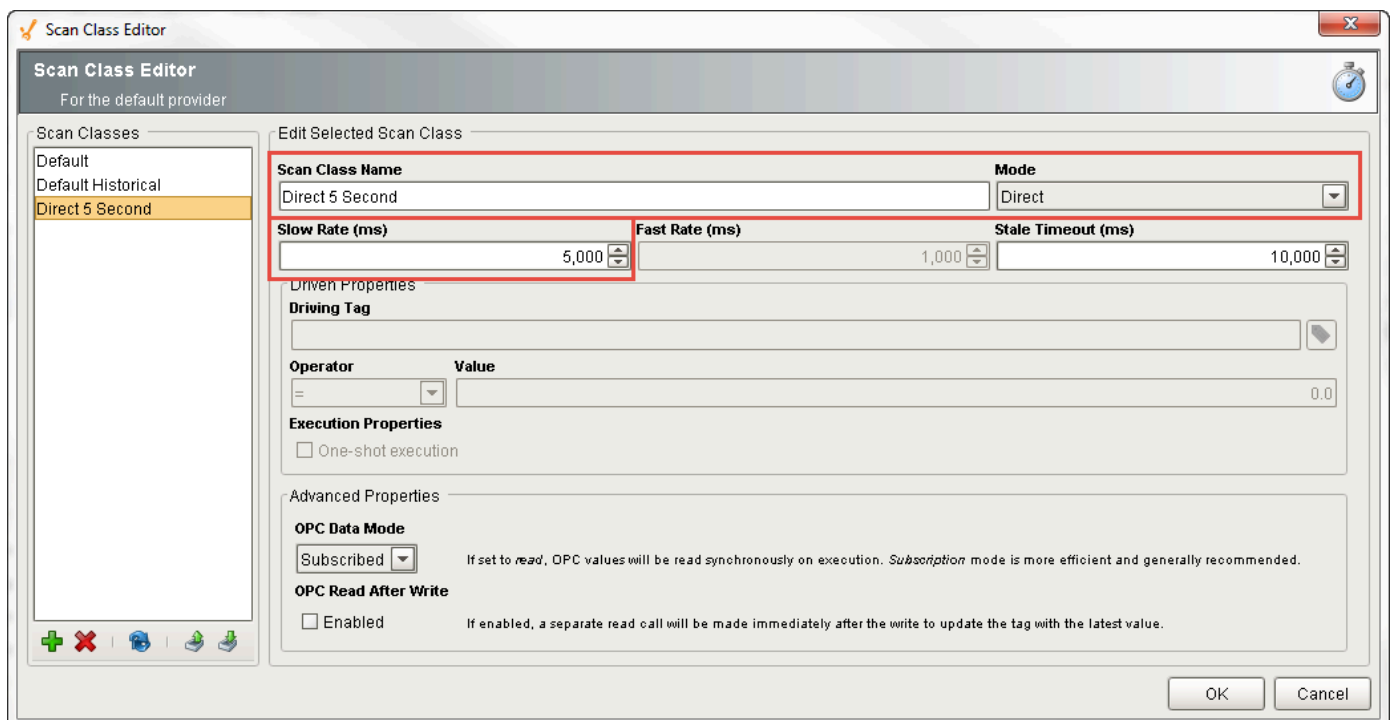
On the left side of the window, you can see the default scan classes: **Default** and **Default Historical**. They are both Direct scan classes. The **Default** scan class polls at a slow rate of 1 second, and **Default Historical** polls at a 10 second rate.



2. To create a new scan class, click the **green +** icon on the bottom-left of the window.

3. Enter the name of your new scan class: **Direct 5 Second**.

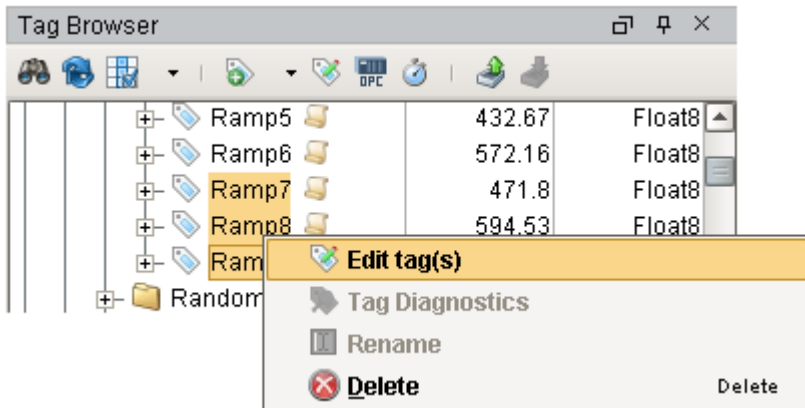
- From the **Mode** dropdown, select **Direct**.
- Set the **Slow Rate (ms)** to the number of milliseconds you want to poll at. This example sets the slow rate at **5,000** ms or 5 seconds.
- Click **OK**.



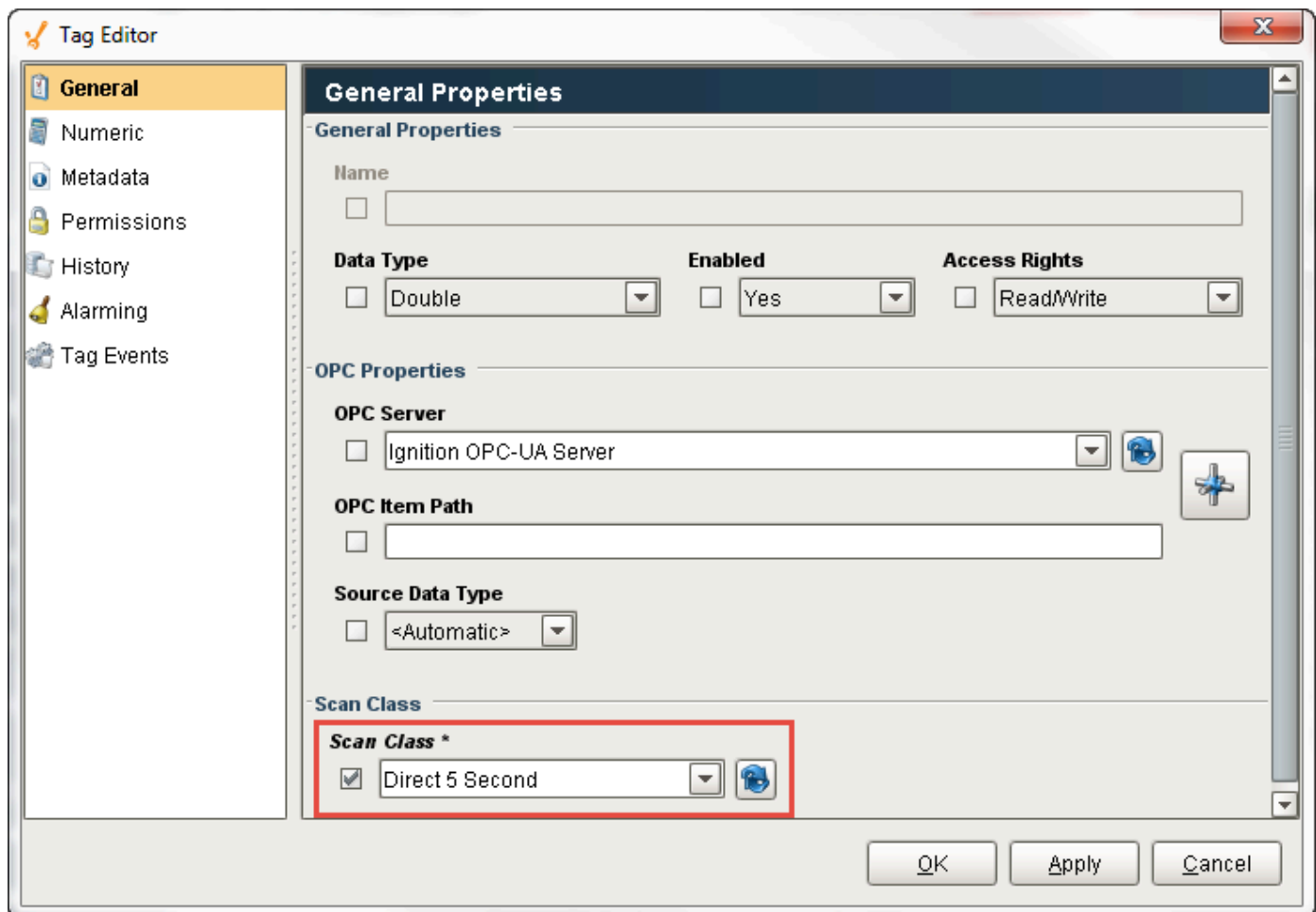
4. Once you have your Direct scan class created, let's add multiple Tags to this scan class.

Go to your **Tag Browser**, find some Tags you want to add to the scan class. This example uses several Ramp Tags.

Right click on the selected Tags, and click on **Edit tag(s)**.



5. This opens the **Tag Editor** window. Select the **Direct** scan class from the dropdown, and click **OK**.



You created a new Direct scan class and added your Tags. Just make sure you want to poll those values all the time (24/7) when you use a Direct scan class.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Driven Scan Class

Driven Scan Classes

The Driven scan class is a very flexible and powerful scan class type in Ignition. It allows Tags to run at a fast or slow rate based on a Tag and some condition. This allows for better system performance, while still giving you high resolution data when needed.

Driven scan classes set the polling rate dynamically for Tags based on a comparison between the driving Tag's value and a value. In simple cases, when the logic comparison is true, the Fast Rate applies, otherwise, the Slow Rate applies. There are two exceptions to this: the Any Change operator and the one-shot mode. Using either of these exceptions, the scan class does not run at a rate. 'Any change' will execute once each time the value changes, and 'one-shot' will execute once when the comparison condition is true, and not again until the condition becomes false and subsequently true. This is not the same as setting the slow rate to 0, which means the values should not be polled at all. Having a rate of 0 can be useful (such as when a line is not running), but make sure your Tags are still polling if you are storing history or have alarms set up.

A driving Tag can be an OPC Tag, or an Expression Tag that performs complex calculations and references other Tags. It's possible to create robust scan class triggering using Expression Tags.

BEWARE THE DRIVING TAG'S SCAN CLASS

Make sure the driving Tag is never using the scan class it is driving. The driving Tag should be set to a separate scan class that runs at an appropriate (fast) rate. Changes to the driving Tag will be recognized by the system faster, and the change in poll rate on the Driven scan class will occur faster.

If the Driving Tag is using a scan class that has a 0 rate, it will never have its value checked, and the scan class it is driving will never update.

We have examples of three common ways to run a Driven scan class: by **Machine State**, **Manual Trigger**, and **Time of Day**. Each has a description of the scan class, how to set it up, as well as how to setup the driving Tag.

Driven Scan Class - Machine State



INDUCTIVE
UNIVERSITY

Driven Scan Class - Machine State

[Watch the video](#)

Driven scan classes that are based on a machine can be very important when you only want to poll values when a machine is ON or OFF. A machine state scan class is when the Tags are polled at one rate with the machine ON, and a different rate when the machine is OFF. You can easily see the rate change in the Designer using this type of condition.

The screenshot shows the 'Scan Class Editor' window for the 'default provider'. On the left, a list of 'Scan Classes' includes 'Default', 'Default Historical', and 'Driven Machine State' (which is selected). The main area is titled 'Edit Selected Scan Class' and contains the following fields:

- Scan Class Name:** Driven Machine State
- Mode:** Driven (dropdown)
- Slow Rate (ms):** 10,000 (spin box)
- Fast Rate (ms):** 1,000 (spin box)
- State Timeout (ms):** 10,000 (spin box)
- Driven Properties:**
 - Driving Tag:** Test/Machine On (text field with a tag icon)
 - Operator:** = (dropdown)
 - Value:** 1.0 (text field)
- Execution Properties:**
 - ☐ One-shot execution
- Advanced Properties:**
 - OPC Data Mode:** Read (dropdown). Note: If set to read, OPC values will be read synchronously on execution. Subscription mode is more efficient and generally recommended.
 - OPC Read After Write:**
 - ☐ Enabled. Note: If enabled, a separate read call will be made immediately after the write to update the tag with the latest value.

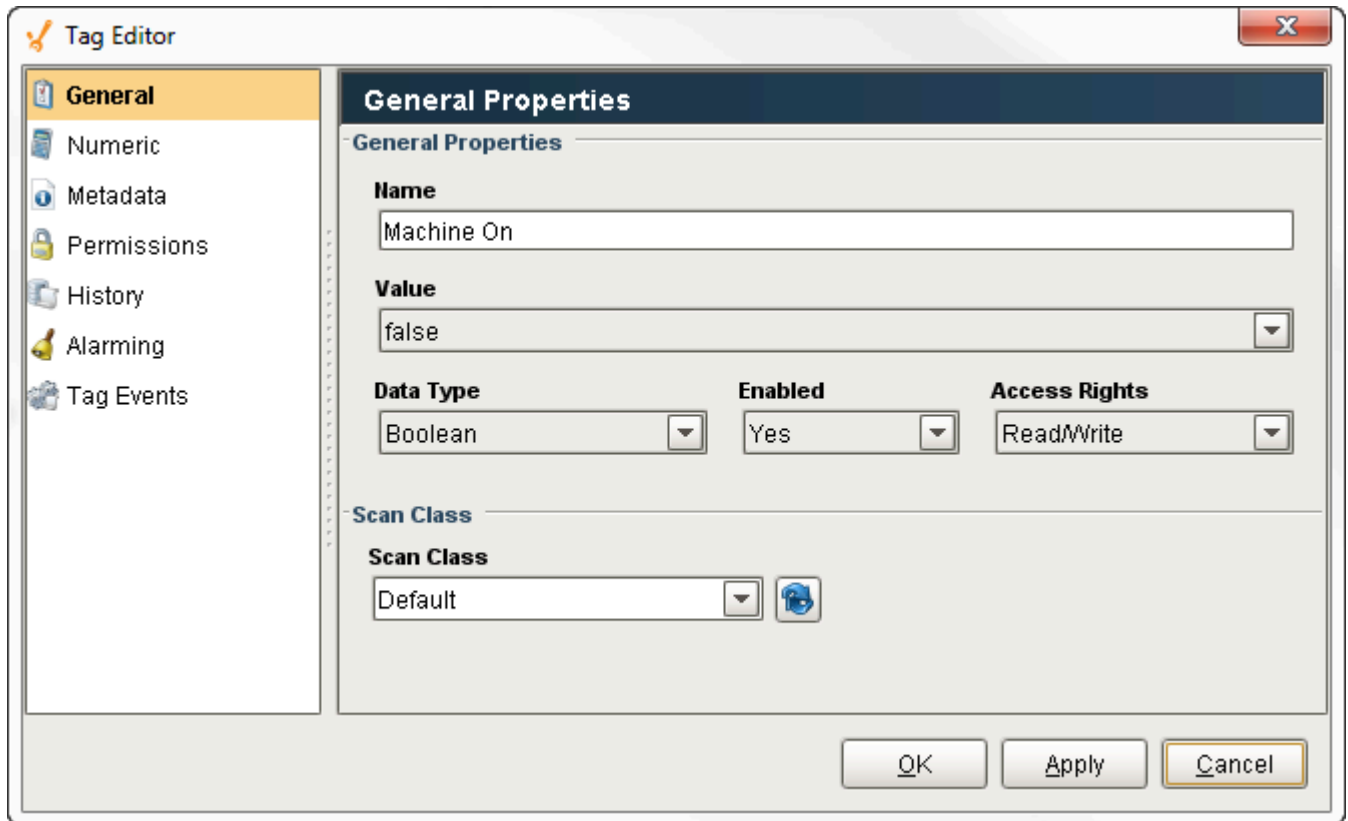
At the bottom right are 'OK' and 'Cancel' buttons.

To Create a Machine State Tag

First, you need to create a driving Tag if you don't already have one.

1. In the **Tag Browser**, right-click on the **Tags** folder, then go to **New Tag > Memory Tag** to **create a memory Tag**. The Tag Editor window is displayed.
2. In **Tag Editor**, enter the following:
 - Name: **Machine On**
 - Value: **false**

- Data Type: **Boolean**



The Tag Editor dialog box is shown with the 'General' tab selected in the left sidebar. The 'General Properties' section contains the following fields:

- Name:** Machine On
- Value:** false
- Data Type:** Boolean
- Enabled:** Yes
- Access Rights:** Read/Write

The 'Scan Class' section contains:

- Scan Class:** Default

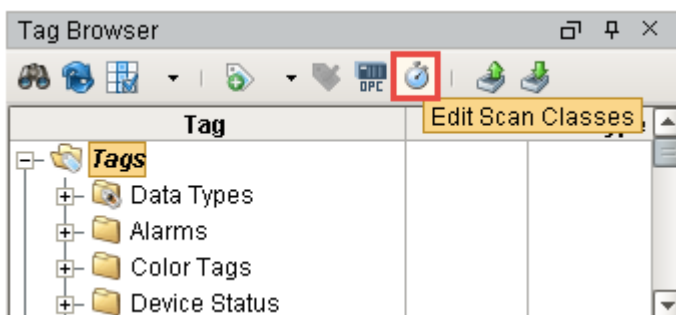
At the bottom right are buttons for OK, Apply, and Cancel.

3. Click **OK** to add the **Machine On** Tag to the Tag Browser.

To Add a Driven Scan Class based on the Machine State

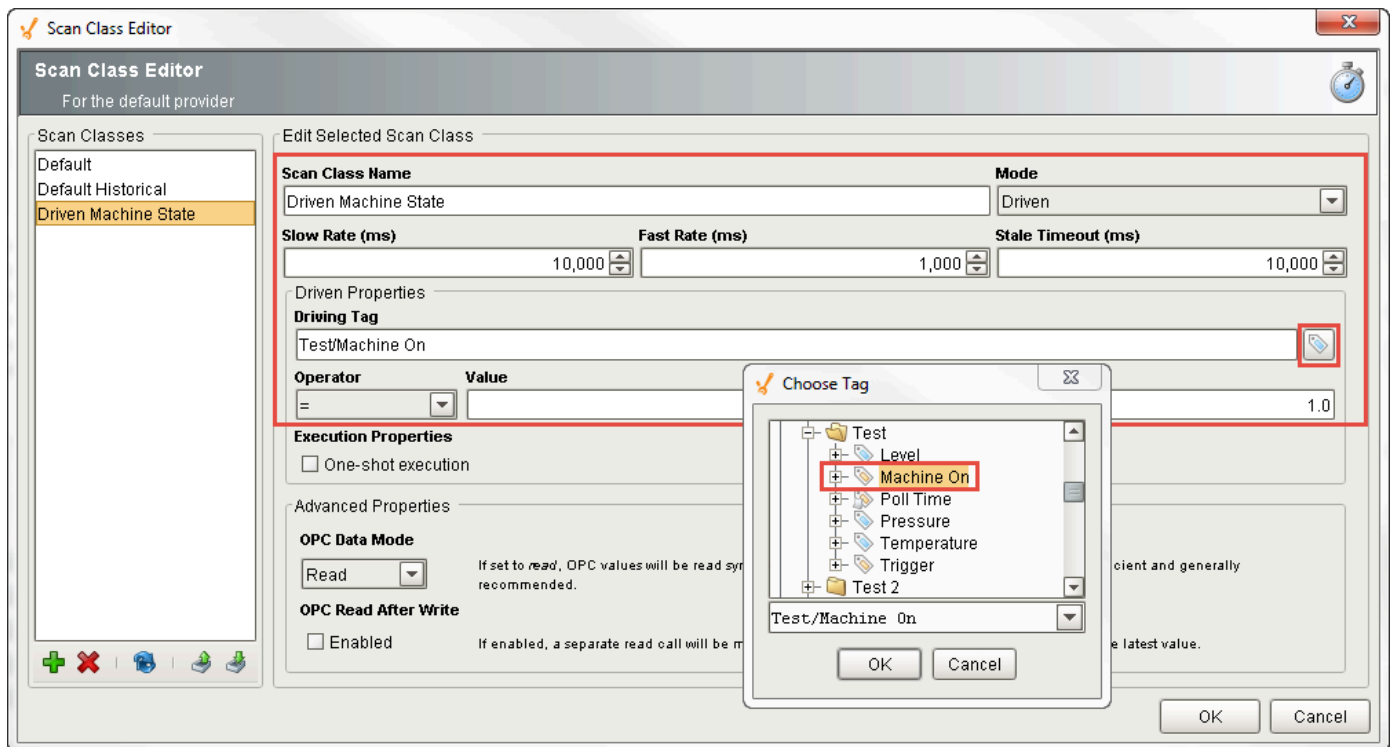
Once you have your driving Tag created, add a Driven scan class that updates the Tags based on when the machine is ON.

1. In the **Tag Browser**, click on the **Timer** icon (⌚) to open the Scan Class Editor.



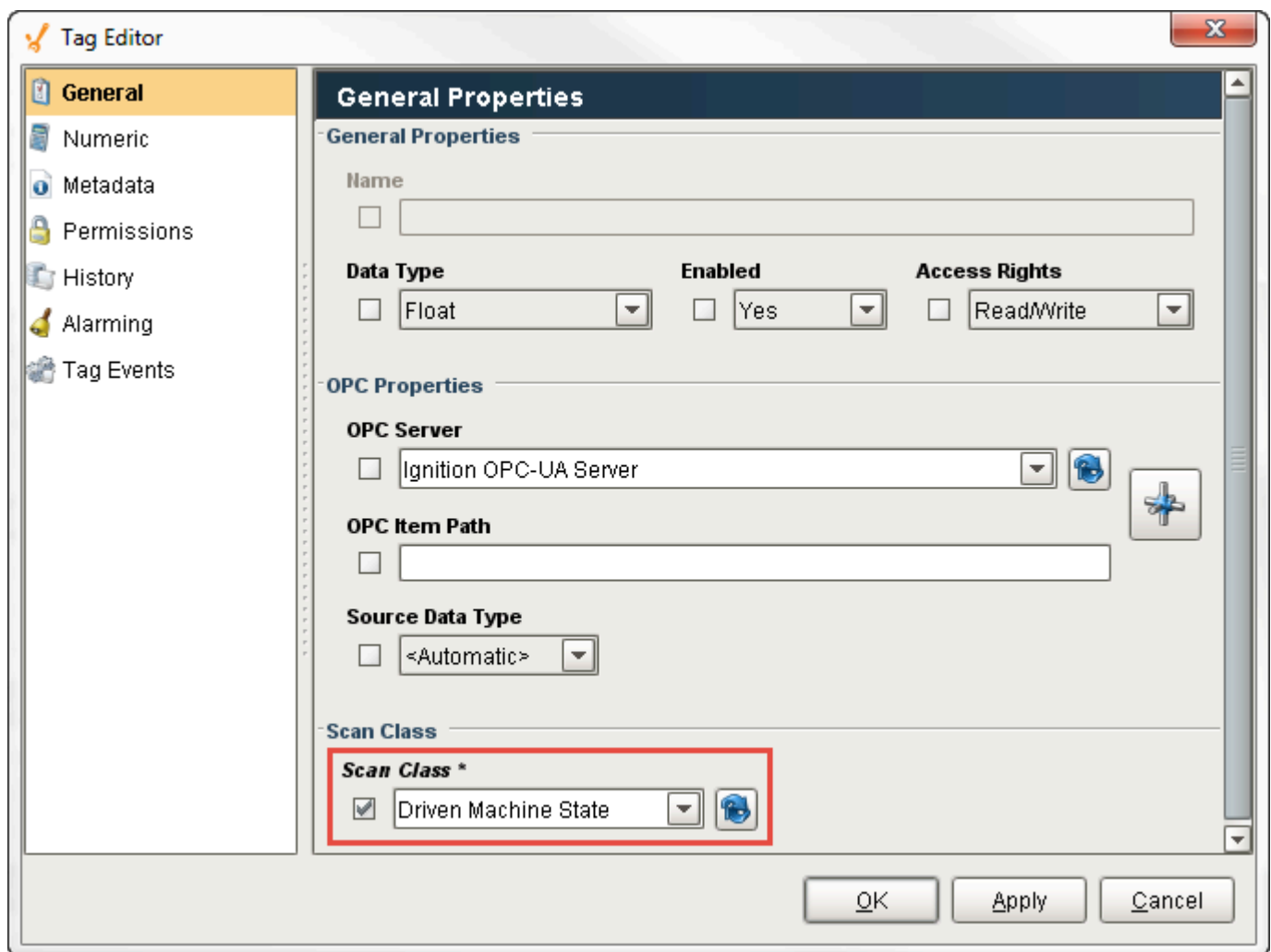
2. To create a new scan class, click on the **green +** icon on the lower-left of the window.
3. Enter the name of the scan class. For example, you can name it **Driven Machine State**, and set the **Mode** to **Driven**. This scan class executes based on the condition you set on the **Driving Tag**.

- Click on the **Tag** icon next to the **Driving Tag** box, and choose the Tag you want to set the condition on, and click **OK** on the popup. This example uses the **Machine On** Tag that was just created.
- Set the **Slow Rate** to 10,000 (10 seconds) and the **Fast Rate** to 1000 (1 second).
- Set the **Operator** property to = and the **Value** property to 1. When the machine is ON it will use the Fast Rate, and when the machine is OFF it will use the Slow Rate. Click **OK** on the Scan Class Editor window.

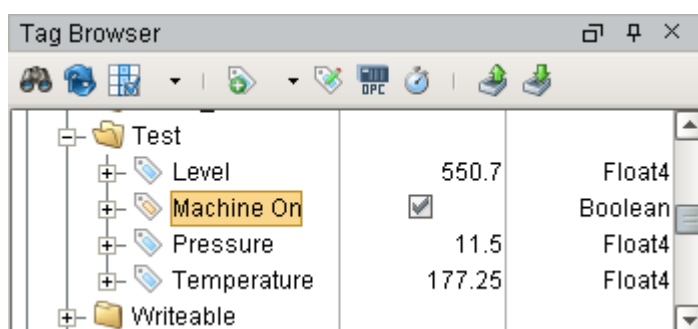


- Next, select all the other Tags that you want to use the scan class for (i.e., **Level**, **Pressure**, and **Temperature**). Right click on one of the selected Tags and choose **Edit Tags**. To change the scan class, choose our new **Driven Machine State** from the Scan Class dropdown, and click **OK**.

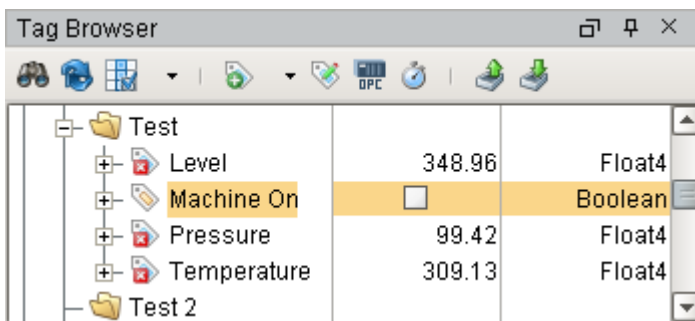
Tag Browser		
Test		
Level	128.89	Float4
Machine On	☐	Boolean
Pressure	77.16	Float4
Temperature	275.74	Float4
Writeable		



8. In the **Tag Browser**, mark the checkbox next to the Tag (i.e., **Machine ON**), that you created to turn the machine ON. Now, when the machine is ON, polling will be at the 1 second rate. When the machine is OFF, polling will be at the 10 second rate. Test it by toggling the Machine Tag ON and OFF.



You can also set the Slow Rate or the Fast Rate to a value of 0, which means polling is stopped when the condition is false. So for example, when the machine is OFF, you will see the Tags stay at the last known value, and a overlay will appear on the affected Tags.



Tag Name	Value	DataType
Level	348.96	Float4
Machine On	☐	Boolean
Pressure	99.42	Float4
Temperature	309.13	Float4

Once the machine is turned back ON, the Tag overlays disappear. The overlay will only appear on the affected Tags one time, and that is when the slow or fast polling rate was run for the first time after initially being set to a value of 0.

Driven Scan Class - Manual Trigger



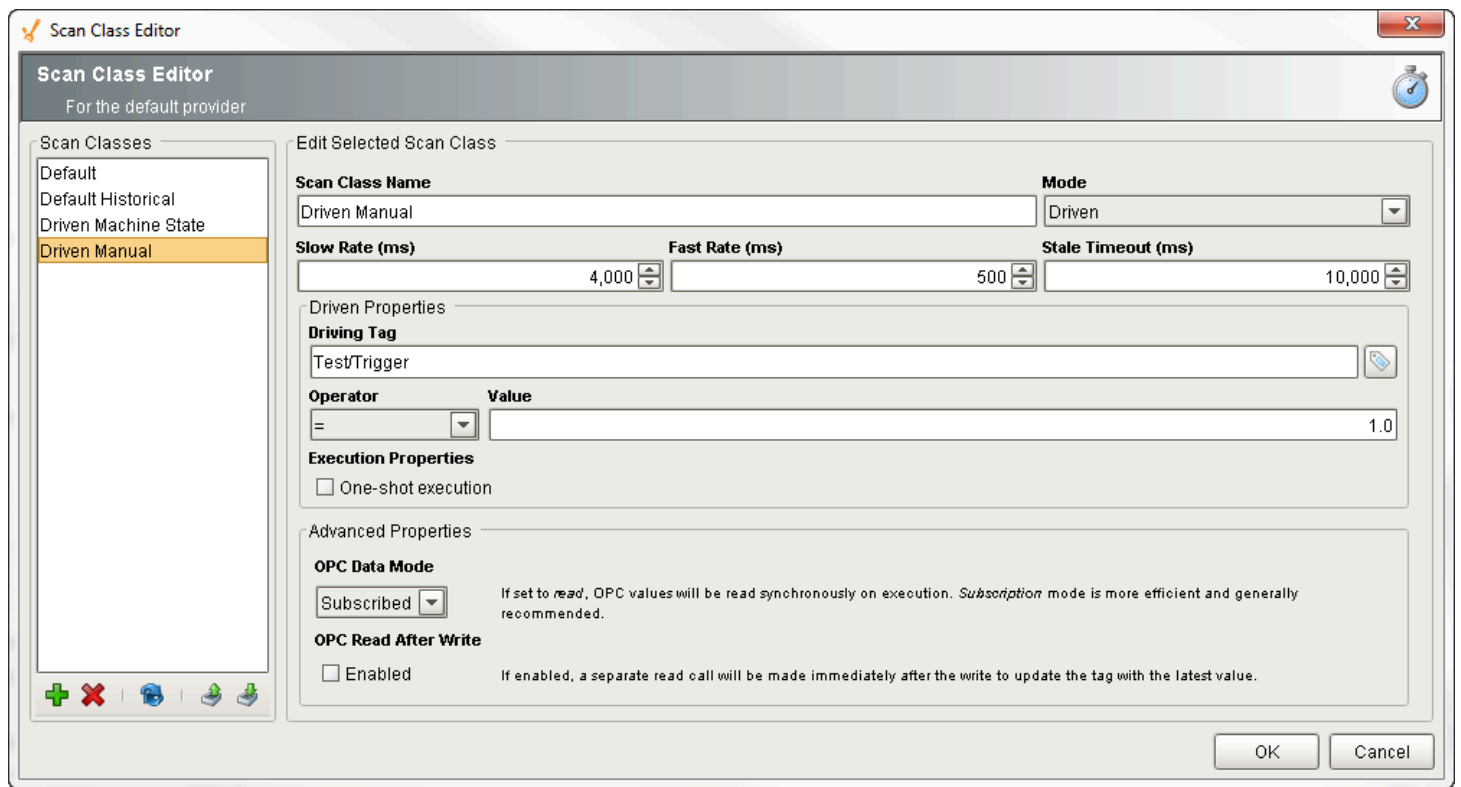
INDUCTIVE
UNIVERSITY

**Driven Scan Class - Manual
Trigger**

[Watch the video](#)

Using a Manual Trigger is another example of a Driven scan class that can be used to allow operators to speed up or slow down polling rates on demand based on a trigger value. A Manual Trigger is similar to the Driven scan class for Machine State, but rather than get the trigger from the PLC, you get the trigger from Ignition. When the Trigger value is equal to 1, it will use the **Fast Rate**, and you can see the values changing very fast. When the Trigger is equal to 0, it will use the **Slow Rate**.

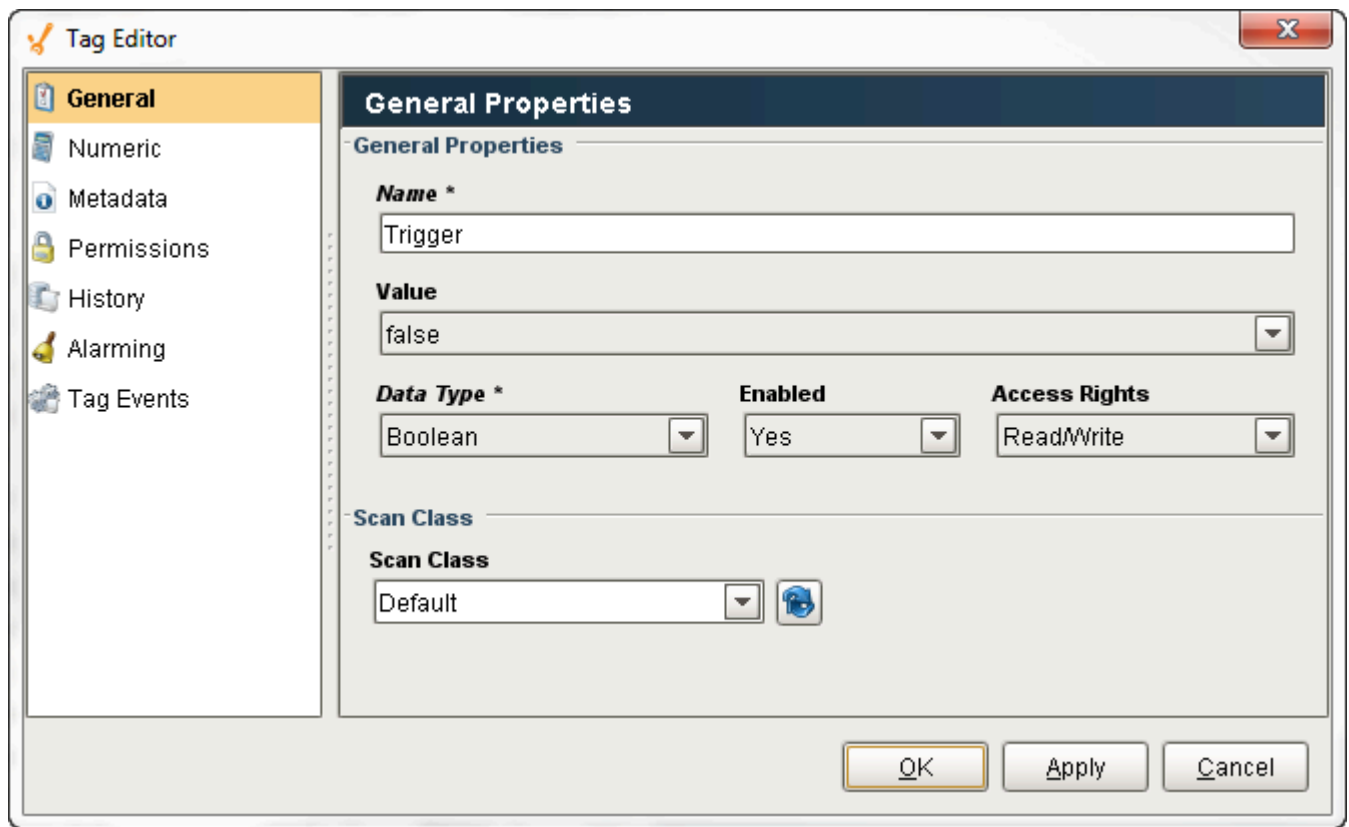
A good example of using a Manual Trigger is where the operator uses a 2-State Toggle Button component on a window to turn the Trigger ON to poll at a fast rate, and then to turn the Trigger OFF to poll at the slow rate whenever necessary.



To Create a Trigger Tag

In order to get the trigger from Ignition, you need to create a driving Tag.

1. In the **Tag Browser**, right-click on the **Tags** folder, then go to **New Tag > Memory Tag** to **create a memory Tag**. The Tag Editor window is displayed.
2. In Tag Editor, enter the following:
 - Name: **Trigger**
 - Value: **false**
 - Data Type: **Boolean**

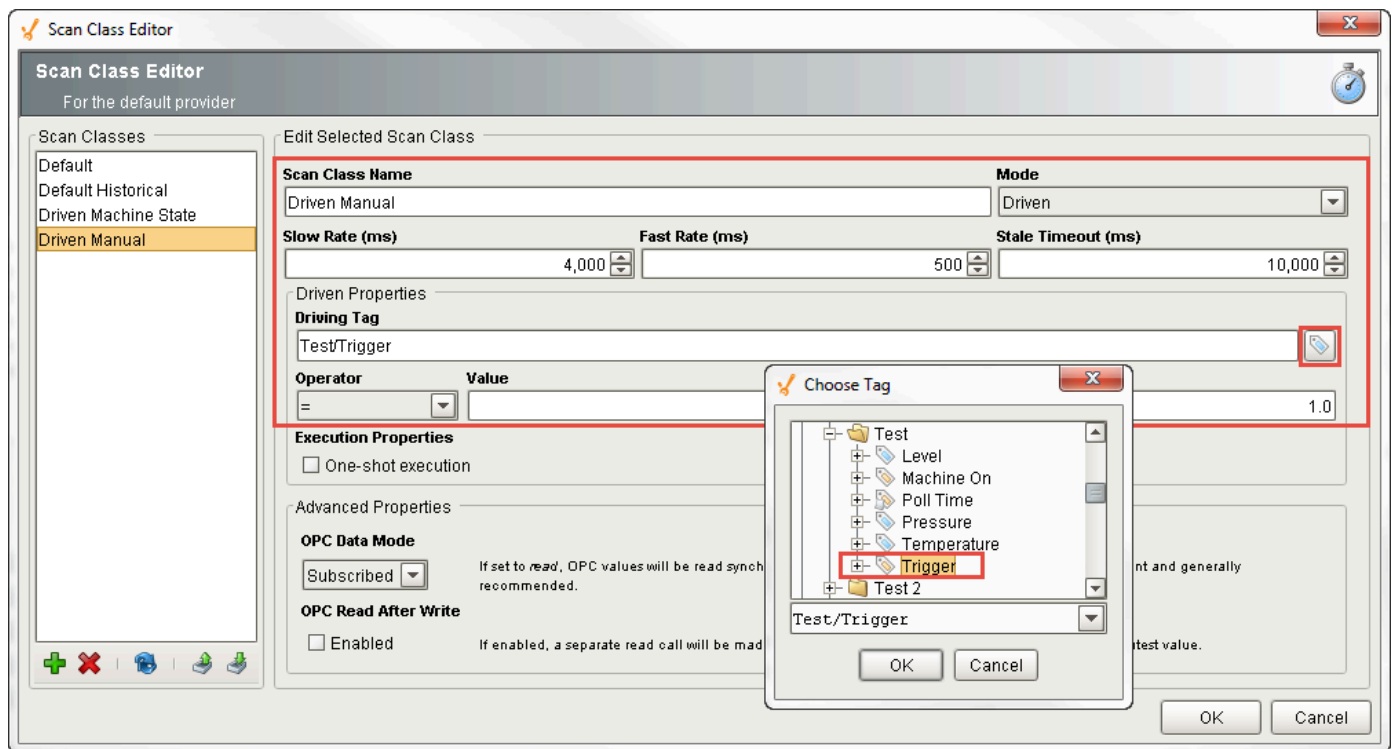


3. Click **OK** to add the **Trigger** Tag to the Tag Browser.

To Add a Driven Scan Class based on a Trigger Value

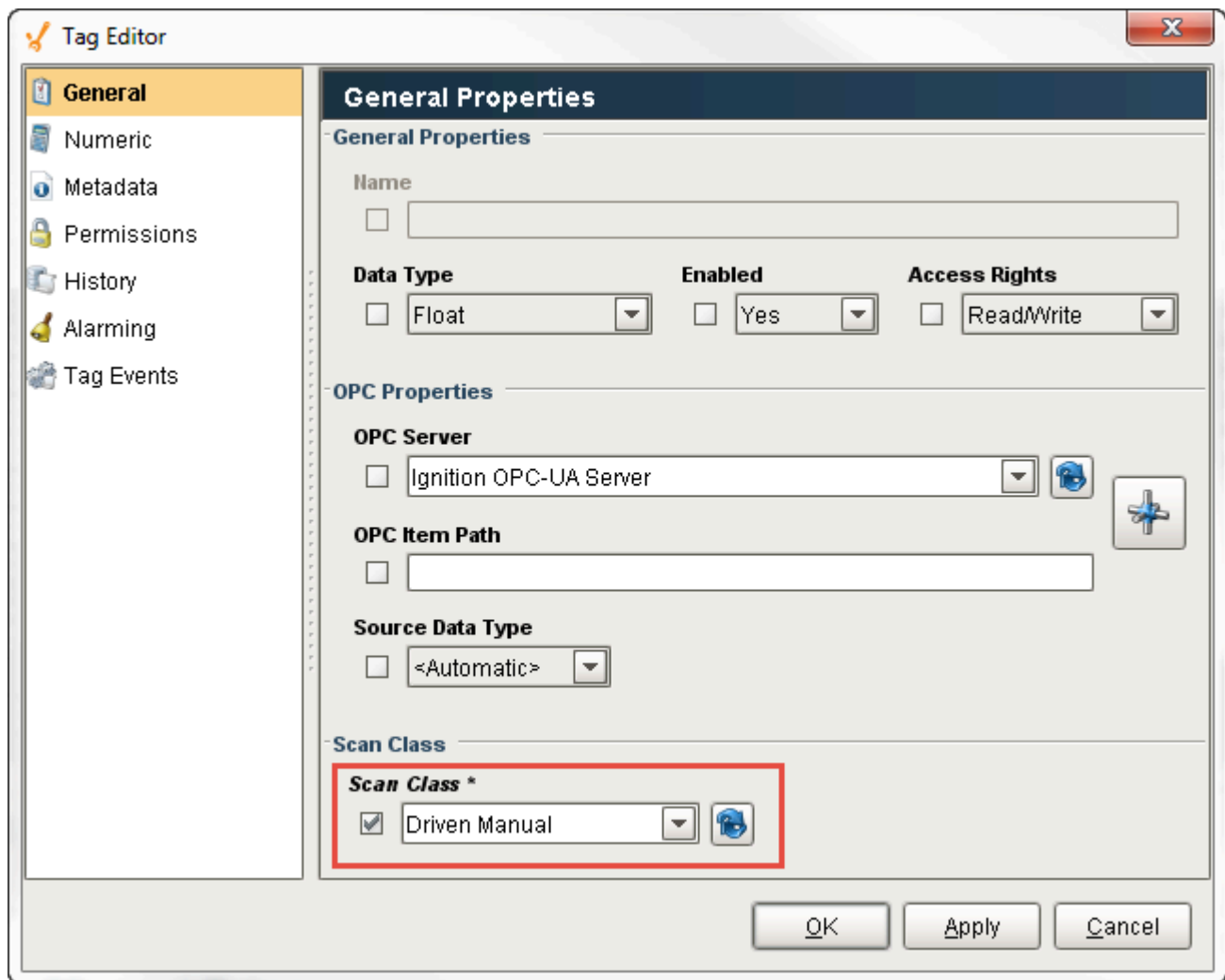
Now that you have your Trigger Tag, add a Driven scan class that changes based on the value of the Trigger Tag.

1. In the **Tag Browser**, click on the **Timer** icon () to open the Scan Class Editor.
2. Create a new scan class, by clicking on the **green +** icon on the lower-left of the window
3. Name the scan class, for example, you can name it **Driven Manual**, and set the **Mode** to **Driven**. This scan class executes based on the condition you set on the **Driving Tag**.
4. Click on the **Tag** icon next to the **Driving Tag** box, and select the Tag you want to set the condition on, and click **OK**. This example uses the **Trigger** Tag that was just created. Click **OK**.
5. Set the **Slow Rate** to **4000** (4 seconds) and the **Fast Rate** to **500** (1/2 second, a very fast rate).
6. Set **Operator** property to **=** and the **Value** property to **1**. When the Trigger is equal to 1 it will use the **Fast Rate**, and when the Trigger is equal to 0 it will use the **Slow Rate**. Click **OK**.



7. Next, select all the other Tags that you want to specify in the scan class (i.e., **Level**, **Pressure**, and **Temperature**). Right click on the selected Tags, and the **Tag Editor** will open. To change the scan class, choose **Driven Manual** from the dropdown, and click **OK**.

Tag Browser		
Test Level Machine On Poll Time Pressure Temperature Trigger Test 2		
Level	45.93	Float4
Machine On	☐	Boolean
Poll Time	☒	Boolean
Pressure	31.07	Float4
Temperature	206.6	Float4
Trigger	☒	Boolean



8. Now, in the Tag Browser you will see a checkbox next to the Trigger Tag which you can set the ON/OFF condition. So when the Trigger is set to ON, you are going to run at the Fast Rate of 500 ms. If the Trigger is set to OFF, it will run at the Slow Rate of 4000 ms.

Tag Name	Value	Data Type
Level	600.74	Float4
Machine On	☐	Boolean
Pressure	39.3	Float4
Temperature	68.95	Float4
Trigger	☒	Boolean

9. Lastly, you can drag the **Trigger** Tag onto a window and select the **Control > Two-state Toggle** button to allow your operators to control the scan class speed.

Driven Scan Class - Time of Day



INDUCTIVE
UNIVERSITY

Driven Scan Class - Time of Day

[Watch the video](#)

Driven scan classes can be used as the polling rate for Tags to trigger at different rates at different times of the day. They can also be used as a one-shot event at a specific time of the day. You can accomplish this by setting the condition of the Driven scan class to be true at certain times of the day, and false at other times.

Keep in mind, that if you use a Driven scan class based on the time of day with a 0 Slow Rate for history, the Tag will only store history during those specified times.

Scan Class Editor
For the default provider

Scan Classes

- Default
- Default Historical
- Driven Machine State
- Driven Manual
- Time Driven**

Edit Selected Scan Class

Scan Class Name: Time Driven Mode: Driven

Slow Rate (ms): 0 Fast Rate (ms): 1,000 Stale Timeout (ms): 10,000

Driven Properties

Driving Tag: TestPoll Time

Operator: = Value: 1.0

Execution Properties

☐ One-shot execution

Advanced Properties

OPC Data Mode: Subscribed

OPC Read After Write: ☐ Enabled

OK Cancel

To Create a Time of Day Tag

Let's use an Expression Tag to drive the condition. You can use the functions that are in the expression language to poll the PLC during the hours of 8am to 5pm, and if the machine is ON. In this example, both of these conditions must be true for the value to poll. This example assumes you already have a **Machine On** boolean Tag in the same folder.

1. In the **Tag Browser**, [create an Expression Tag](#) that is a Boolean. Right click on the Tag folder then go to **New Tag > Expression**. The Tag Editor window is displayed.

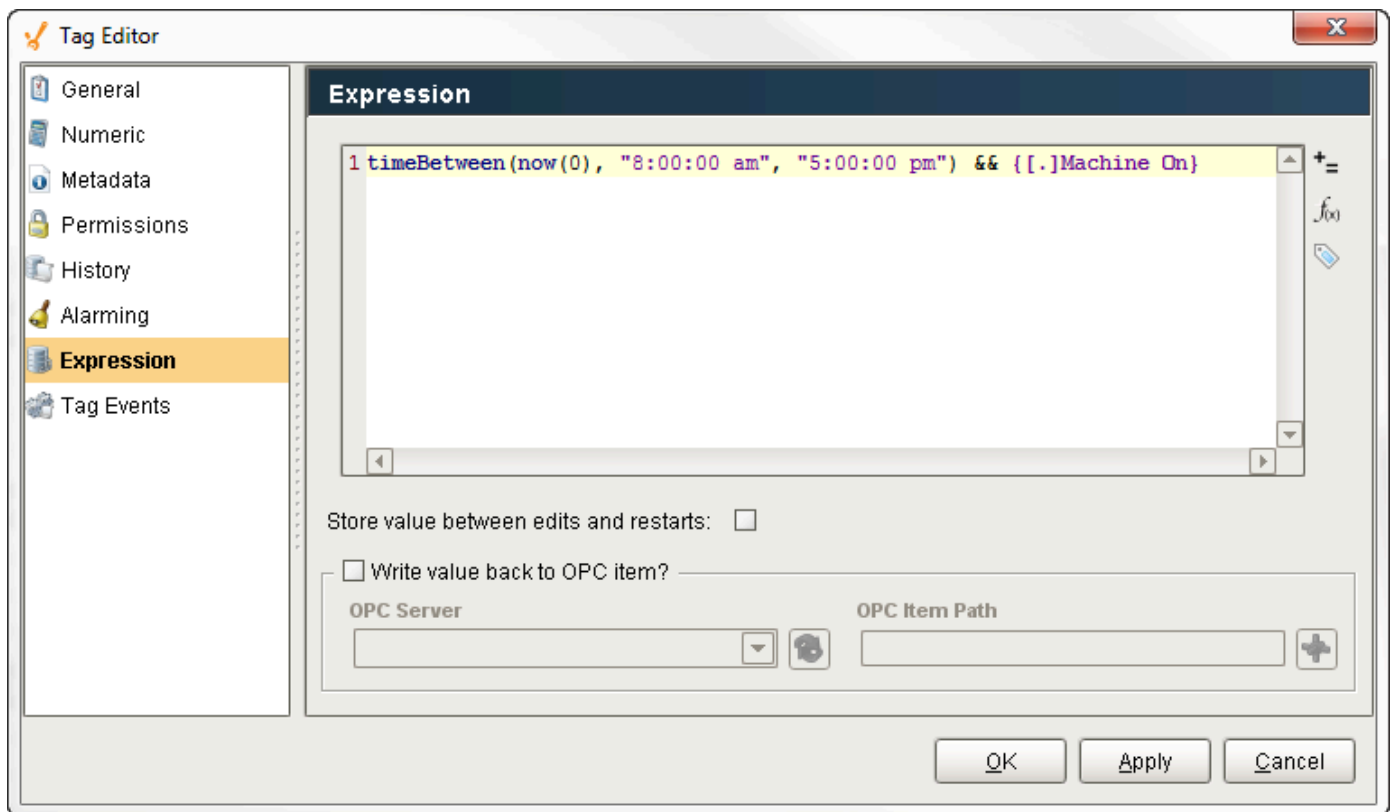
Enter the following:

- Name: **Poll Time**

- Value: **false**
 - Data Type: **Boolean**
2. Click on **Expression** from the menu on the left, and enter the expression below. The condition is true when the time is between 8am and 5pm, and the machine is ON, otherwise it is false at other times, and/or when the machine is OFF. Click **OK**.

timeBetween and Machine On expression

```
timeBetween(now(0), "8:00:00 am", "5:00:00 pm") && {[.]Machine On}
```

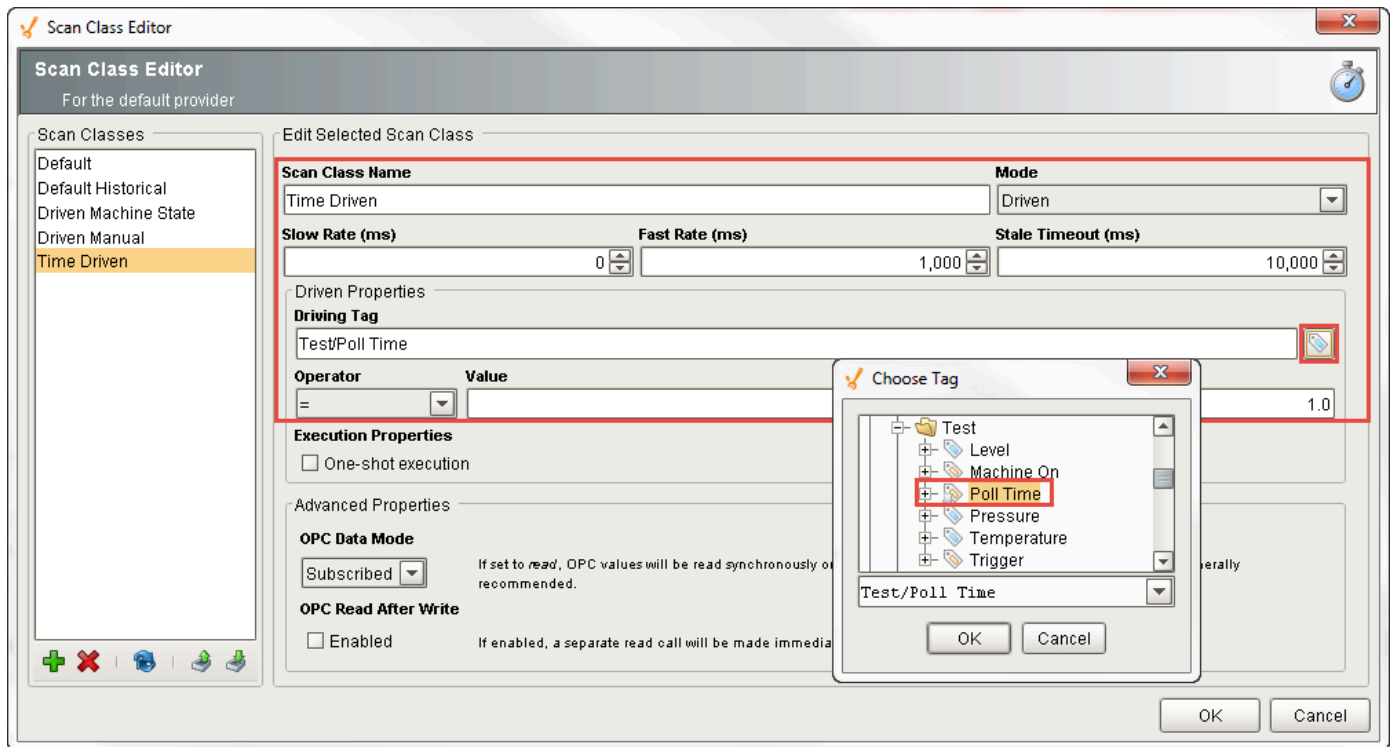


To Add a Driven Scan Class based on the Time of Day

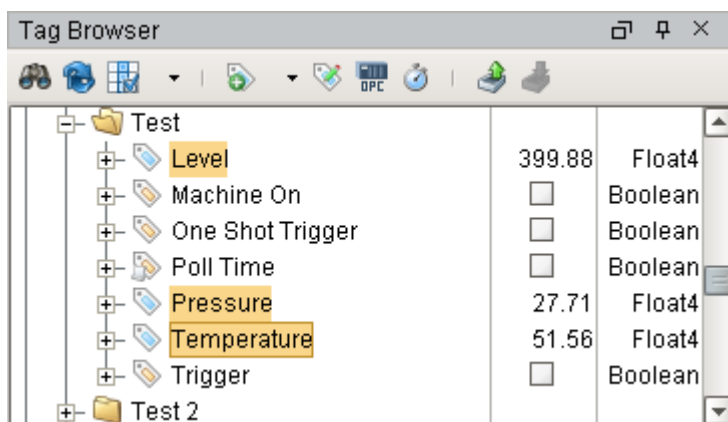
Now that you have your Poll Time Tag created, add a Driven scan class that updates the Tags based on a time of day and when the machine is ON using the expression.

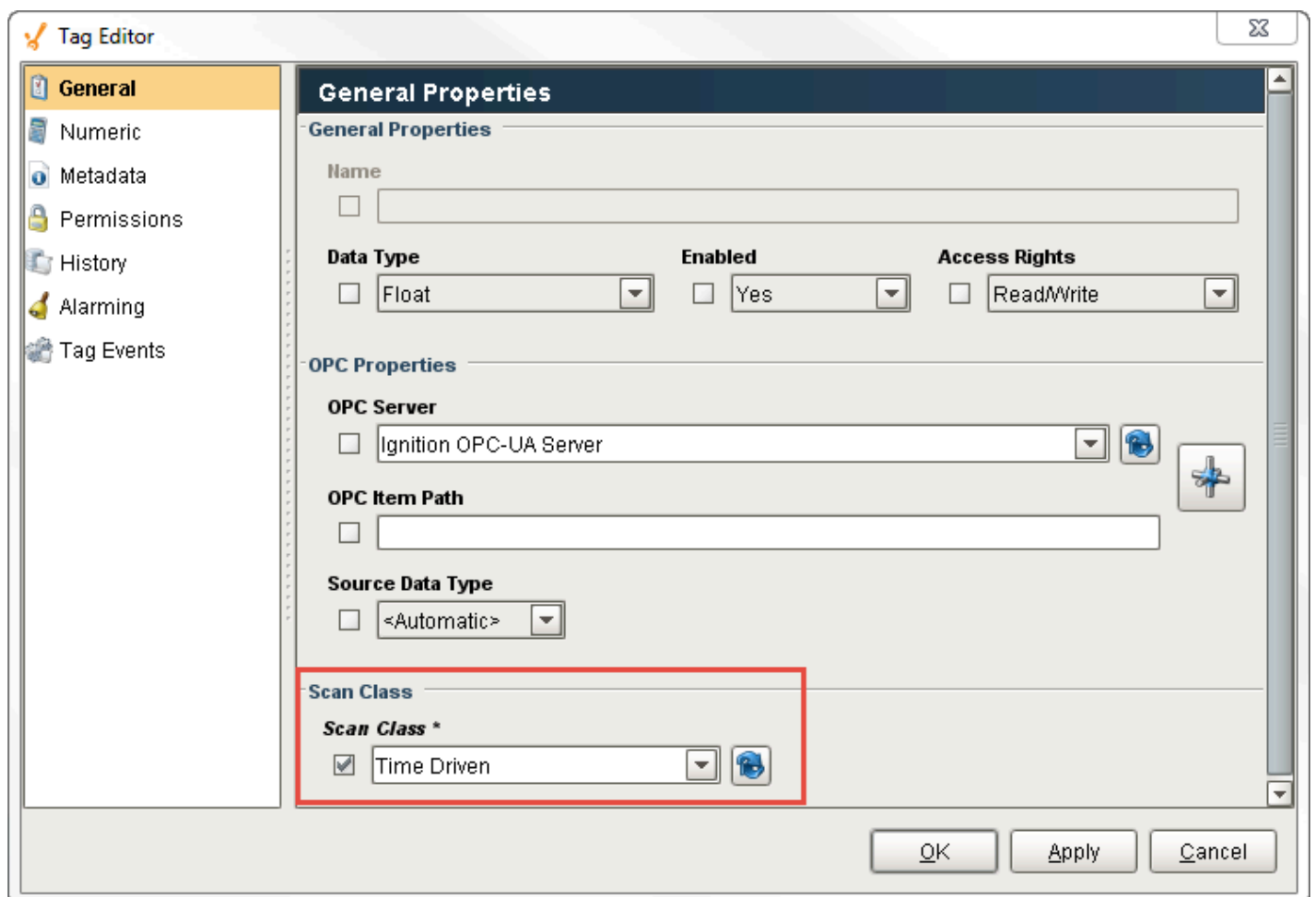
1. In the **Tag Browser**, click on the **Timer** icon () to open the Scan Class Editor.
2. To create a new scan class, click on the **green +** icon on the lower-left of the window
3. Enter the scan class name to **Time Driven**, and set the **Mode** to **Driven**. This scan class executes based on the condition of the **Driving Tag**.

4. Click on the **Tag** icon next to the **Driving Tag** box, and select the Tag you want to set the condition on, and click **OK**. This example uses the **Poll Time** Tag that was just created.
5. Set the **Slow Rate** to **0** so you don't poll the condition when it is false. Set the **Fast Rate** to **1000ms**.
6. Set the **Operator** property to **=** and the **Value** property to **1.0**. Click **OK**.

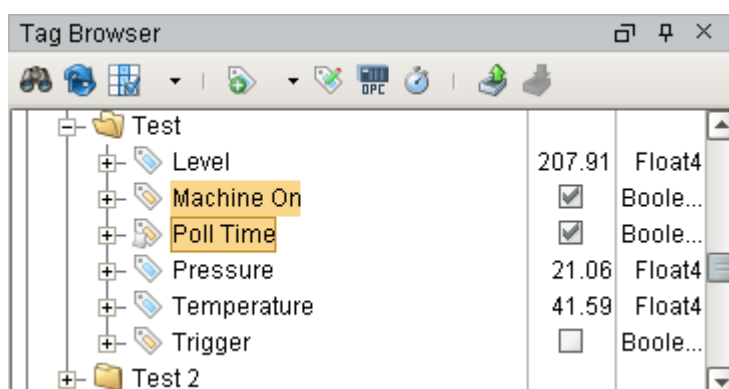


7. Next, select all the other Tags that you want to specify in the scan class (i.e., **Level**, **Pressure** and **Temperature**). Make sure you don't select the Machine On Tag since it is helping to drive the Scan Class. Right click on the selected Tags, and the **Tag Editor** will open. To change the scan class, choose **Time Driven** from the dropdown, and click **OK**.





8. In the Tag Browser you will see checkboxes next to the Poll Time and Machine On Tags. When the driving conditions are true, that is the time between hours of 8am and 5pm, and when the machine is ON, polling is at the Fast Rate of 1000 ms. When both conditions are false, polling is stopped, and the last known value is displayed.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Driven Scan Class - One Shot



INDUCTIVE
UNIVERSITY

Driven Scan Class - One Shot

[Watch the video](#)

The Driven scan class can be configured as a One-shot scan class. This will not run at a rate, but instead it will be triggered by a change in the driving Tag's value. One-shot polling executes once when the condition becomes true, then waits for that condition to become false then true again. This makes it follow a 'rising edge' or 'falling edge' pattern.

Let's make a Driven One-shot scan class that only updates once when a Memory Tag goes to true. We are using a Memory Tag so a user can change the value when they want to poll the PLC. This is actually a good example of how an operator can poll the PLC for new values whenever they want to, using a refresh button on the screen.

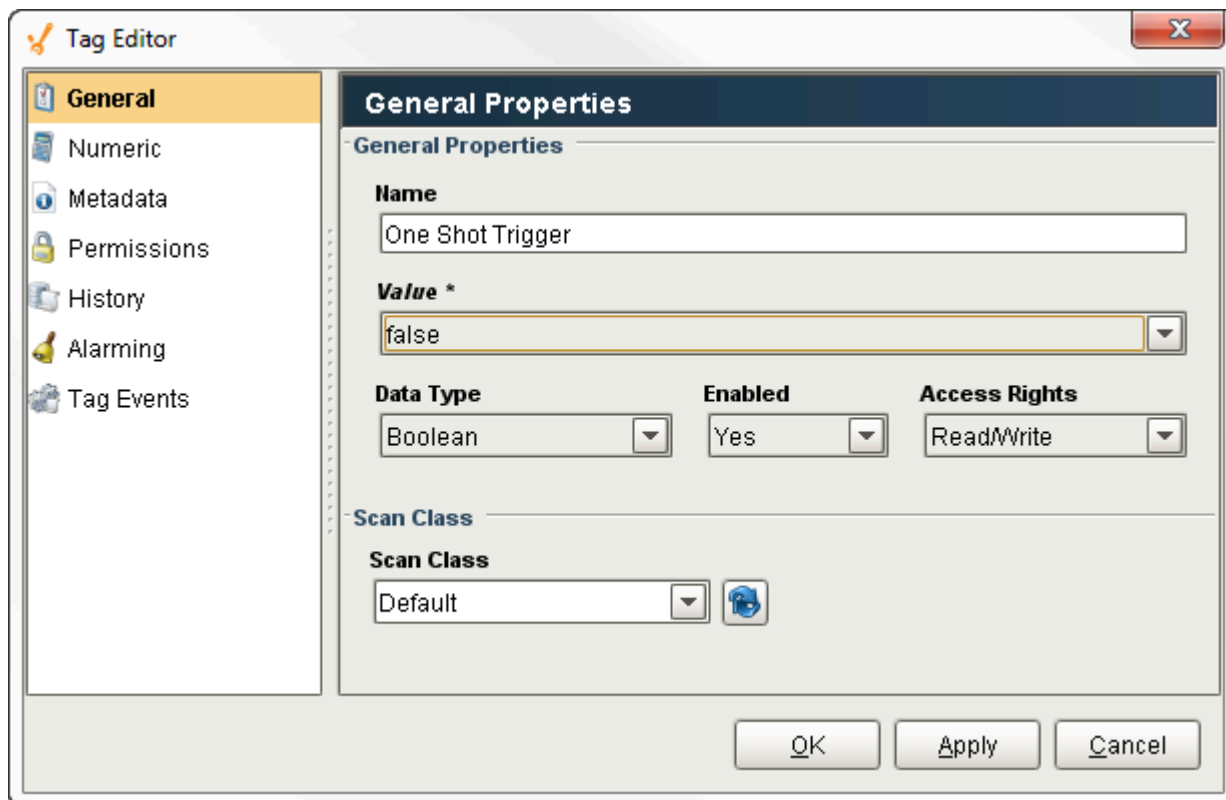
To Create a Memory Tag

1. In the **Tag Browser**, right click on the Tags folder, then go to **New Tag > Memory Tag** to create a memory Tag.

The Tag Editor will open.

2. In the **Tag Editor**, enter the following:

- Tag Name: **One Shot Trigger**
- Value: **false**
- Data Type: **Boolean**



3. Click **OK** to add the One Shot Trigger Tag to the Tag Browser.

To Make a Driven One-shot Scan Class

Once you have your driving Tag created, add a Driven - One shot scan class that lets a user poll Tags manually.

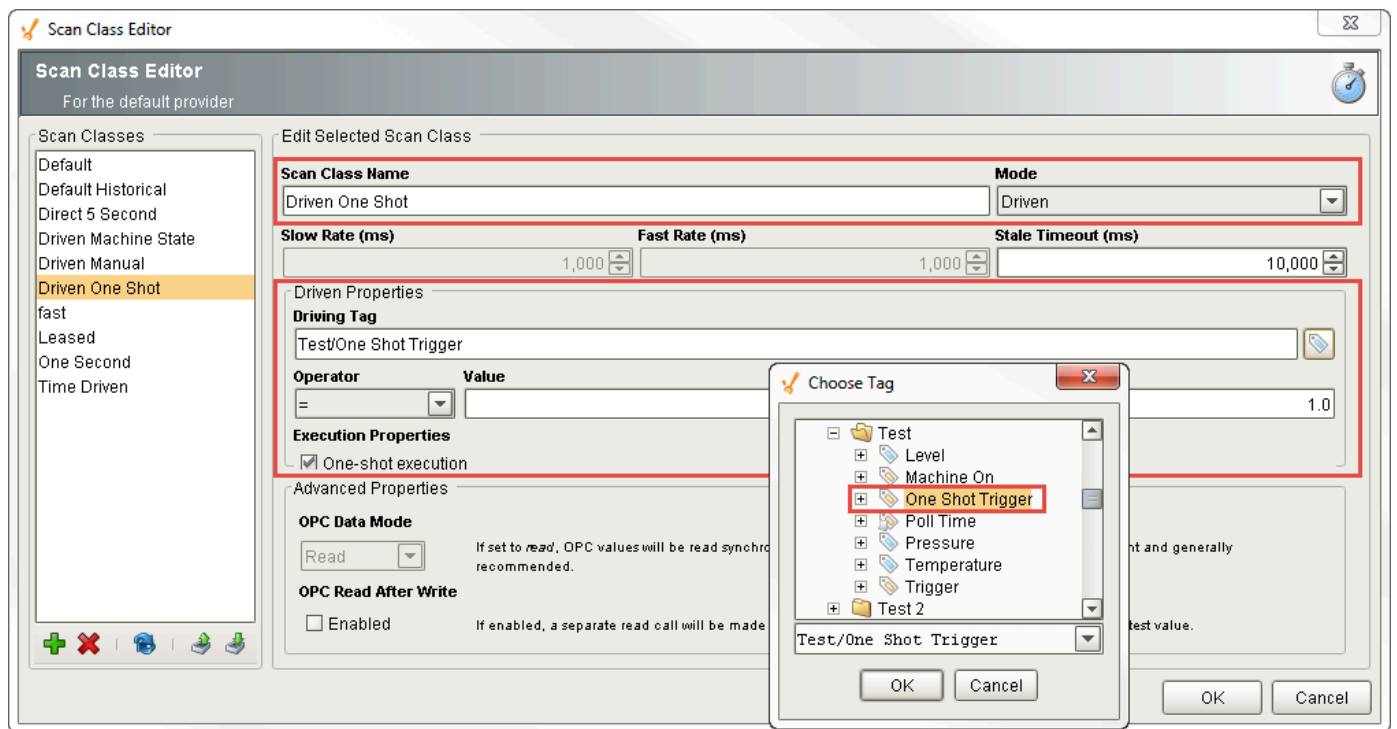
1. In the **Tag Browser**, click on the **Timer** icon () to open the Scan Class Editor.
2. Create a new scan class by clicking on the **green +** icon on the lower-left of the window.
3. Enter the name of the scan class. For example, you can name it **Driven One Shot**, and set the **Mode** to **Driven**.

This scan class executes based on the condition you set on the **Driving Tag**.

Ignore the slow and fast rate since they are not used.

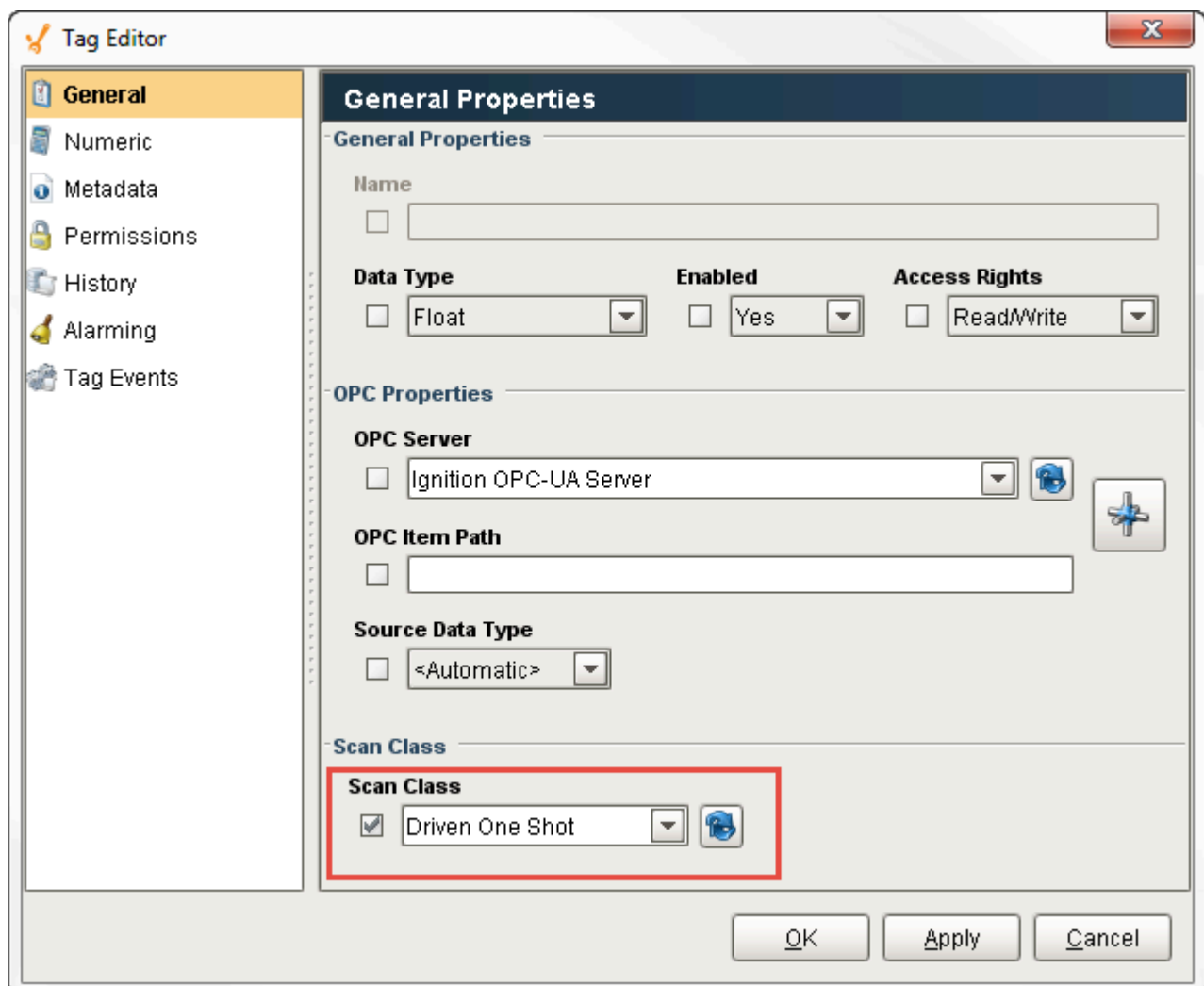
4. Click on the **Tag** icon next to the **Driving Tag** box, and choose the Tag you want to set the condition on, and click **OK**. This examples uses the **One Shot Trigger** Tag that was just created.
5. Set the **Operator** property to **=** and the **Value** property to **1**.
6. Check the **One-shot Execution** property.

Click **OK**.



7. Next, select all the other Tags that you want to run in this scan class (i.e., **Level**, **Pressure**, and **Temperature**).

Right click on the selected Tags, and the **Tag Editor** will open. To change the scan class, choose **Driven One Shot** from the dropdown, and click **OK**.



Tag Browser			
Test Level Machine On One Shot Trigger Poll Time Pressure Temperature Trigger Test 2			
Level	620.41	Float4	
Machine On	☐	Boolean	
One Shot Trigger	☒	Boolean	
Poll Time	☐	Boolean	
Pressure	50.23	Float4	
Temperature	235.35	Float4	
Trigger	☐	Boolean	

8. In the **Tag Browser**, check the One Shot Trigger Tag to set the value to **true**, and you will see the values update once, and then stop.

Uncheck the One Shot Trigger Tag, to set the value to **false**, and check it again to set it to **true**, and you will see the values update again.

Tag Browser			
Test Level Machine On One Shot Trigger Poll Time Pressure Temperature Trigger Test 2			
Level	71.92	Float4	
Machine On	☐	Boole...	
One Shot Trigger	☐	Boole...	
Poll Time	☐	Boole...	
Pressure	45.51	Float4	
Temperature	78.26	Float4	
Trigger	☐	Boole...	

9. Drag a **Momentary Button** component onto a window, then drag your **One Shot Trigger** Tag onto the button. Now your operators can request new values and the button automatically resets the Tag.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Leased Scan Class



INDUCTIVE
UNIVERSITY

Leased Scan Class

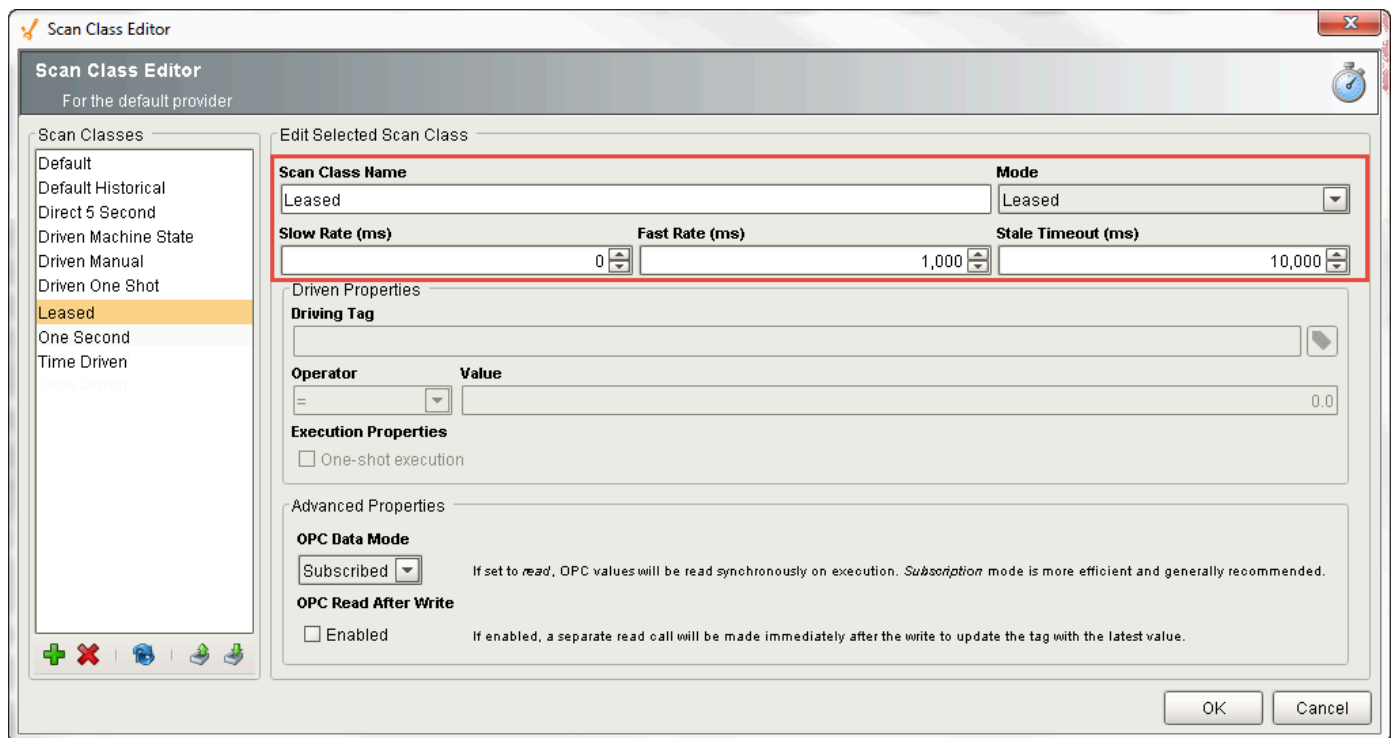
[Watch the video](#)

The Leased scan class allows you to poll Tags at a fast rate when they are subscribed to and visible in the a Client window, or in the Designer. If the Tags are not subscribed to, or if the window in the Client is closed, or the folder containing the Tags in the Designer is not expanded to show the Tags, the scan class runs at the slow polling rate. Essentially, a Leased scan class polls Tags only when someone needs to look at the Tag values.

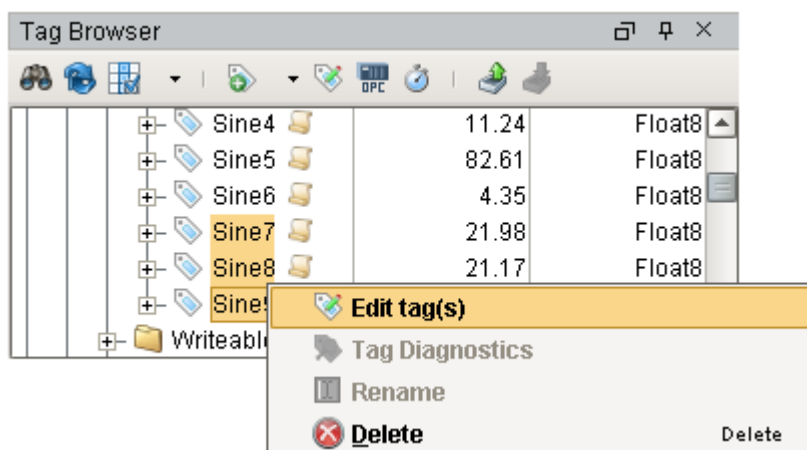
To Add a Leased Scan Class

Let's add a new Leased scan class that polls a Tag from the PLC at a 1 second rate when someone needs to view that Tag in the Client. If the Client is closed, or the Tags are not displayed in the Tag Browser of the Designer, the Tags will not poll at all.

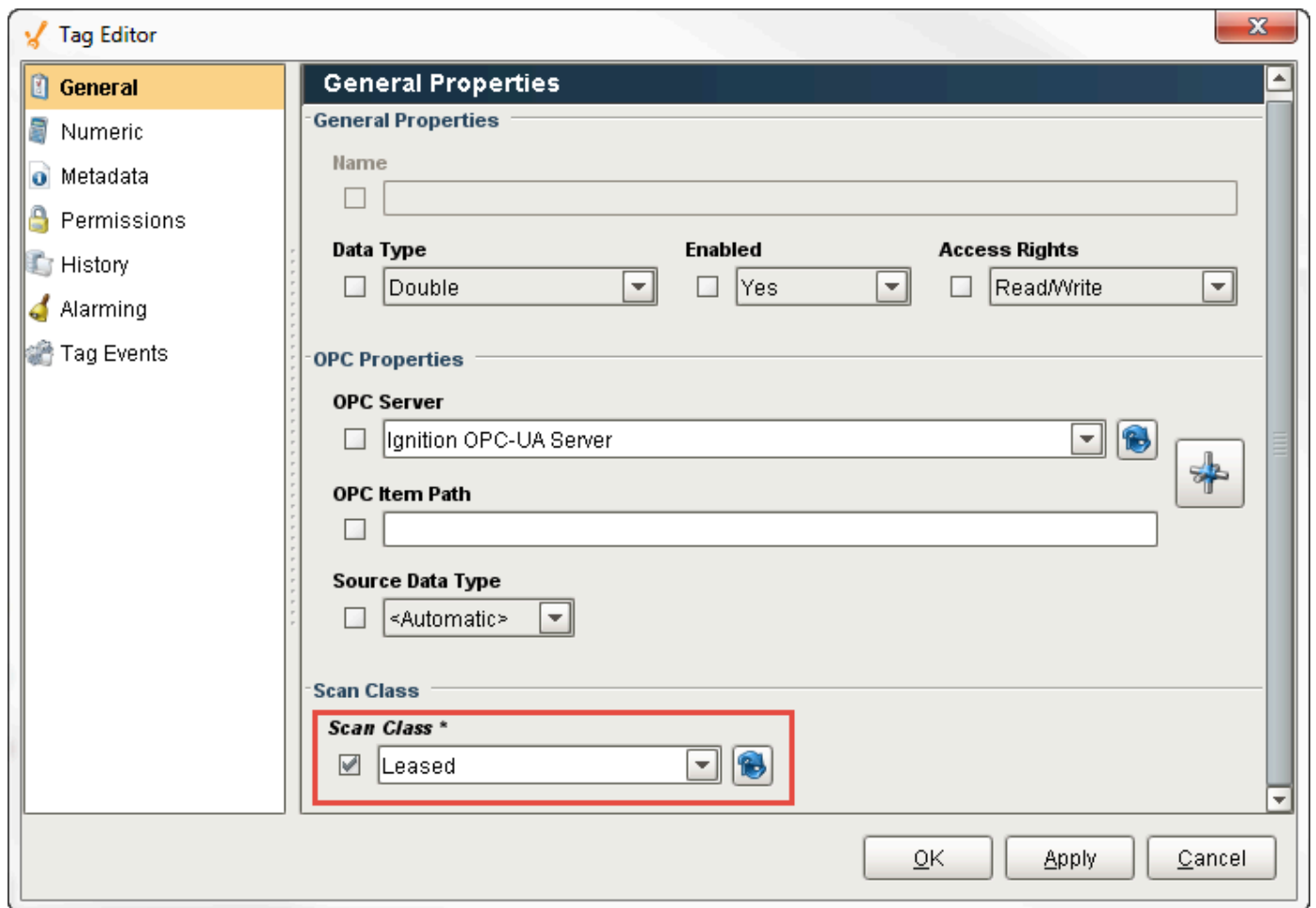
1. Open the **Designer**. In the **Tag Browser**, click on the **Timer** () icon to open the **Scan Class Editor**.
2. On the left side of the window, you can see all existing scan classes. To create new scan class, click on the **green** + icon to add a new scan class.
3. Enter the name of the scan class. For example, name it **Leased**.
 - Set the **Mode** to **Leased**.
 - Set the **Slow Rate** to **0**, and the **Fast Rate** to **1000**, or 1 second.
 - Click **OK**.



- Now, set the scan class on the Tags to use this new **Leased** scan class. Go to your **Tag Browser**, find some Tags you want to add to the scan class. This example uses three Sine Tags. Right click on the selected Tags, and click on **Edit tag(s)**.



- This opens the **Tag Editor** window. Select the **Leased** scan class from the dropdown, and click **OK**.



Now, you just created a new Leased scan class and added your Tags. These Tags are only polled when a component that is bound to it is showing in a Client, or when the Tags are showing in the Designer.

CAUTION

Remember that Tag History is stored based on the current Ignition Tag value. If your Leased scan class has a 0 slow rate, you will not get updated data to store for history.

Always make sure your scan class slow speed is at least as fast as your History scan class speed.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

System, Client, and Diagnostic Tags

System Tags

System Tags provide status about the Ignition system, such as memory usage, performance metrics, and so on. There are Tags for both the Client and the Gateway, and the scope for each is slightly different. You can modify the Gateway system Tags to perform alarming, history, and scaling, but you cannot modify the Client tags. In the **Tag Browser**, under the **System** folder, you can see several **Client System Tags** and **Gateway System Tags** (Not to be confused with the Client Tags).

You can use these Tags in a number of ways, like **changing a client's behavior depending on the IP address** it was opened on.

Tags		
Data Types		
New Tag		Boolean
System		
Client		
Network		
System		
User		
Gateway		
Alarming		
Database		
OPC		
Performance		
Redundancy		
Sessions		
Tags		
CurrentDateTime	2017-09-28 1:50:15 PM	DateTime
SystemName	Test gateway	String
Timezone	America/Los_Angeles	String
Mobile		
Location		
RemoteAddr		String
UserAgent		String
Client		
All Providers		

Client System Tags exist in the client scope and will have different values for each open client. They contain information such as:

- Client IP Address
- Client Hostname
- Client OS
- Who is logged in and what roles they have
- Which project is opened
- Gateway's address
- **New in 7.9.7** Date Formats that change with the Locale
- and more...

Gateway System Tags exist in the Gateway scope and will have the same values for each open client. They contain information such as:

- Database connection information
- Ignition performance stats
- Redundancy information
- Number of client sessions opened
- **New in 7.9.7** The number of seconds the Gateway has been running.
- and more...

Mobile System Tags exist in the Client scope and will have different values for each open client. These tags are unique to systems that have a mobile module, and should be used in conjunction with the Client System Tags for information about a mobile client. They contain information such as:

- Mobile specific services like location.
- and more...

Client Tags



INDUCTIVE
UNIVERSITY

Client Tags

[Watch the video](#)

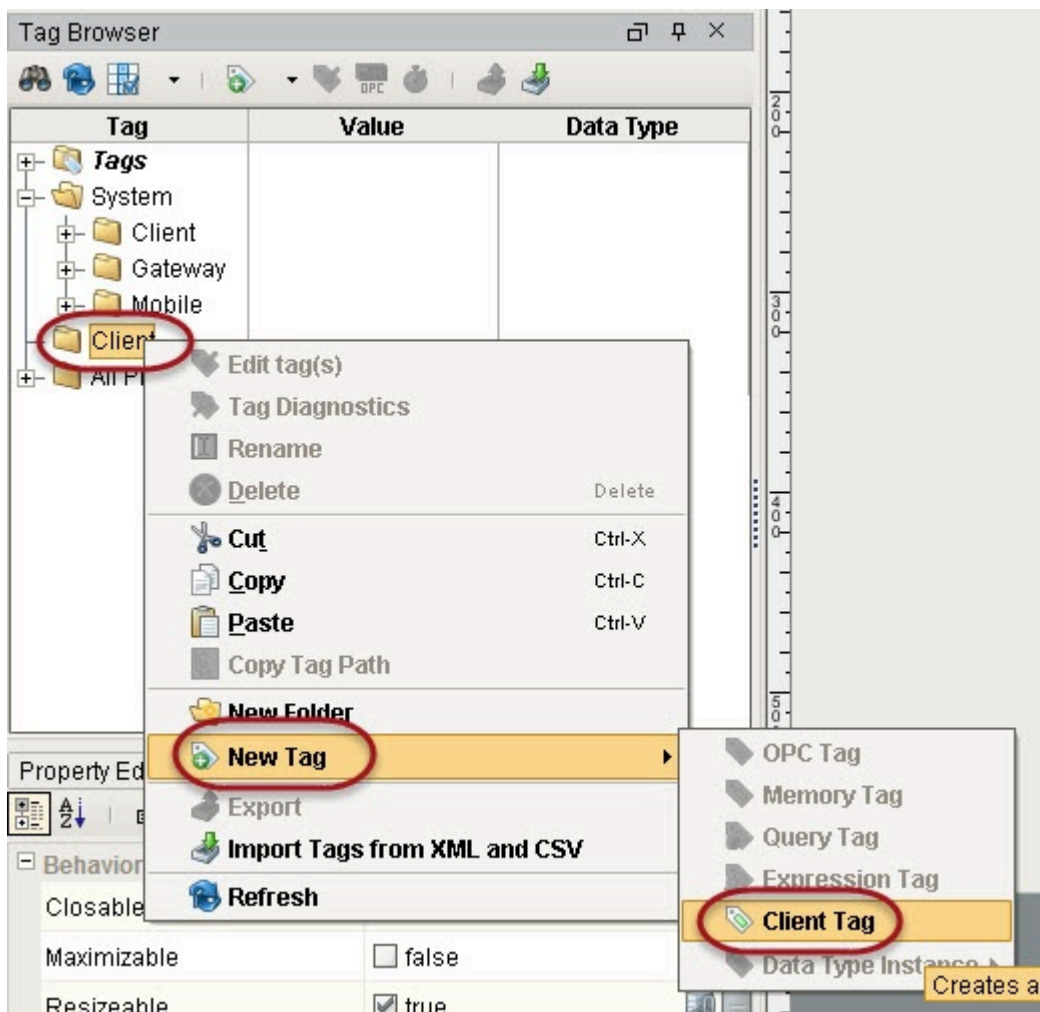
Client tags, as the name implies, are only available for use in the Clients and have a Client scope. This means that their values are isolated to a Client runtime. Even though they are created in the Designer, each Client creates their own instances. This makes them very useful as in-project

variables, for passing information between screens and between other parts of the Clients, such as scripting. Client tags are a hybrid of memory, expression, and SQL Query Tags that you can select between. However, they do not have a scan class and will only poll if you have an expression that forces it to or a SQL Query.

Often, users will create parameterized windows that take in a line ID or machine name. A Client Tag can **store this value for indirection** to be used across multiple windows without users on different clients fighting over the current value as they would with a standard Tag.

To Create a Client tag

1. From the **Tag Browser**, right-click on the Client folder and select **New tag > Client Tag**.



The **Tag Editor** is displayed.

2. In the **Tag Editor**, for example, you can enter the following:

- Name: **Area**
- Value: **Processing**
- Data Type: **String**

3. Click **OK**.

A new tag called **Area** is created in the **Client** folder.

Diagnostics Tags



INDUCTIVE
UNIVERSITY

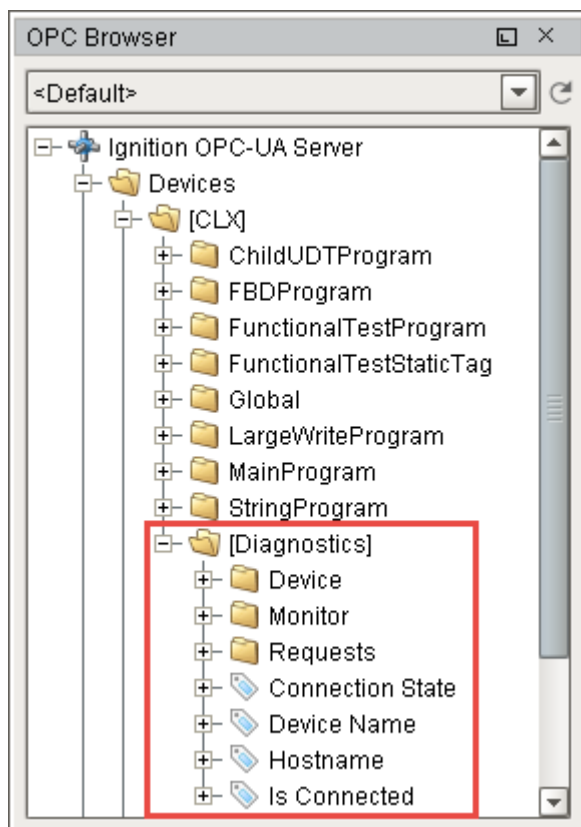
Device Diagnostic Tags

[Watch the video](#)

Every device connection in Ignition comes with a folder of Tags that provide diagnostic information about that device. It contains values for if the device is connected, the last time values were read, or last time a write occurred, and performance information about the device.

You add these tags into the Tag Browser the same way you add Tags from the actual PLC, that is through the **OPC Browser**. Under each device, you can see a special folder called **[Diagnostics]**. You can drag any of the [Diagnostics] Tags or the whole folder into the **Tag Browser** to see live values.

In the [Diagnostics] folder, the most commonly used tag is the **Is Connected** Tag because it is a simple boolean that tells whether Ignition is connected to the device or not. You can easily use this Tag to **show if a PLC is connected** in your project.



This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

System Tags in a Startup Script



INDUCTIVE
UNIVERSITY

Custom Security

[Watch the video](#)

An example of using System tags to determine which window to open given a specific hostname or IP Address:

Normally, security is handled according to the users roles, however sometimes it is important to control access by the location of the Client computer. For example, Clients with certain IP addresses can have a certain level of access, but Clients from computers with different IP addresses can have limited access. Likewise, the computer's host name is also a option.

This practice makes use of the system tags located in these location:

```
[System]Client/Network/Hostname  
[System]Client/Network/IPAddress
```

Each Client will have access to distinct hostname and IP addresses. These can be used in Client startup scripts that evaluate the Client's information and compares it to a list of acceptable host names or IP addresses. This information can come from a database or a set of tags.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

For documentation on the latest stable release, see the [latest version](#) (8.3).

Client Tags for Indirection



INDUCTIVE
UNIVERSITY

**Using Client Tags for
Indirection**

[Watch the video](#)

Client tags can be used as a variable across all windows to indirectly point to a set of tags, such as an area of the plant. Suppose that a window has a drop-down that allows a user to select different areas of the plant. If the drop-down is bidirectionally bound to a Client tag's selected string value property, the user can change the drop-down's value, therefore resulting in the indirect binding throughout the project reflecting the change in the Client tag.

The Client tag can be bound to a Custom property on a window's Root Container. This Custom property can be bound to the Client tag. The components inside the window can have indirect binding on their properties that leverage the Custom property on the window. Therefore, the components on the window will be dependent on the Client tag. Client tags are managed inside the Client therefore each Client will be independent of other Clients.

This is documentation for **Ignition 7.9**, which is no longer actively maintained.

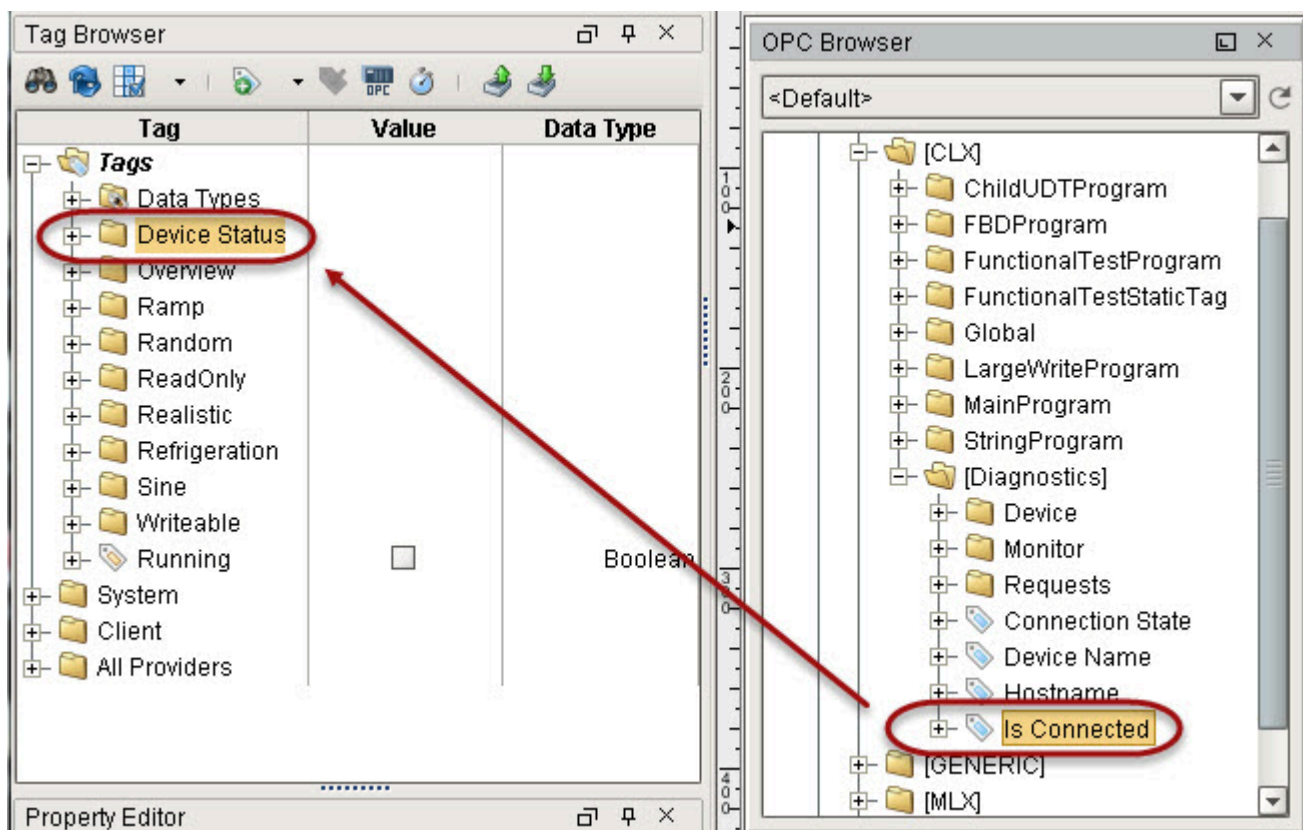
For documentation on the latest stable release, see the [latest version](#) (8.3).

Device Diagnostic Tags to Show PLC Status

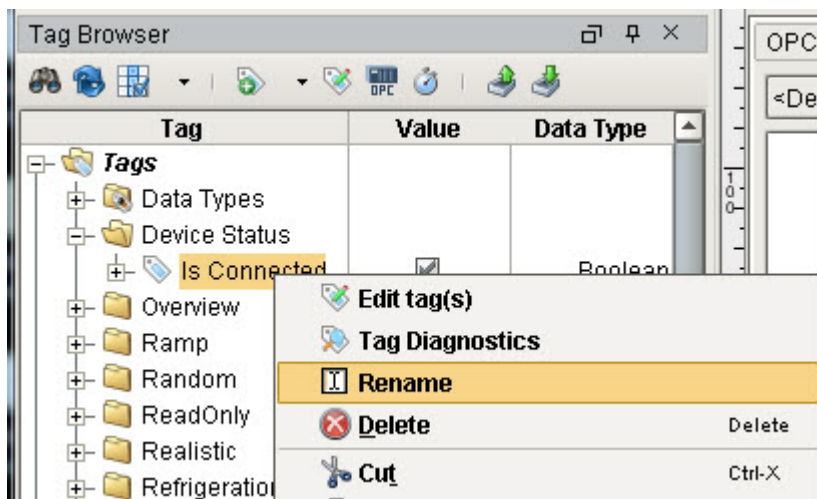
To use a diagnostic tag to show the PLC status

Let's use the **Is Connected** tag from the **CLX** device into the **Device Status** folder.

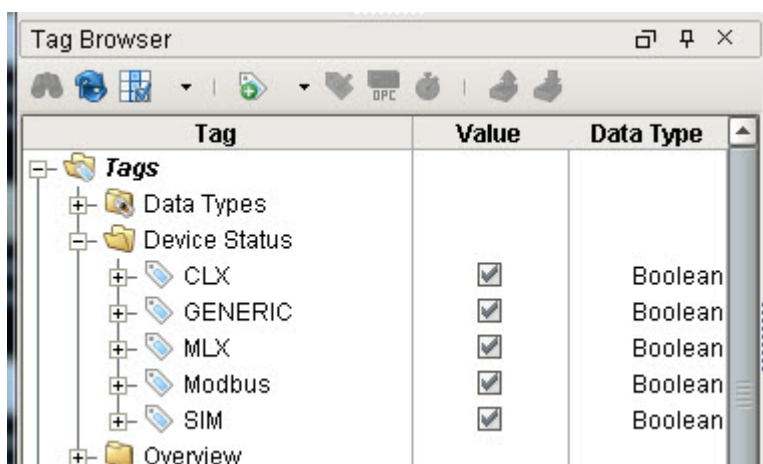
1. Create a folder in **Tag Browser** to contain the status bits by right-clicking on the **Tags** folder and selecting **New Folder**.
2. Name the folder **Device Status** and click **OK**.
3. Drag in the **Is Connected** tag from the **CLX** device into the **Device Status** folder.



4. Right-click on the **Is Connected** tag and rename it to **CLX**.

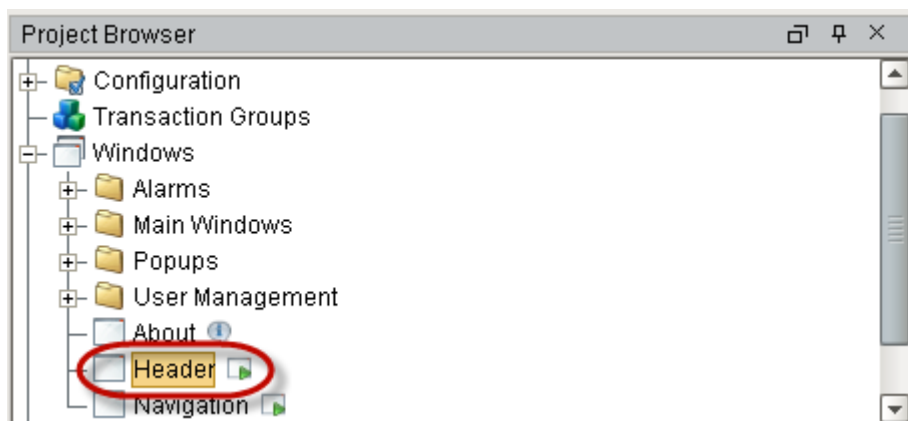


5. Repeat steps 3 and 4 for all device connections you have in Ignition. If you have 5 devices, you will have 5 tags.



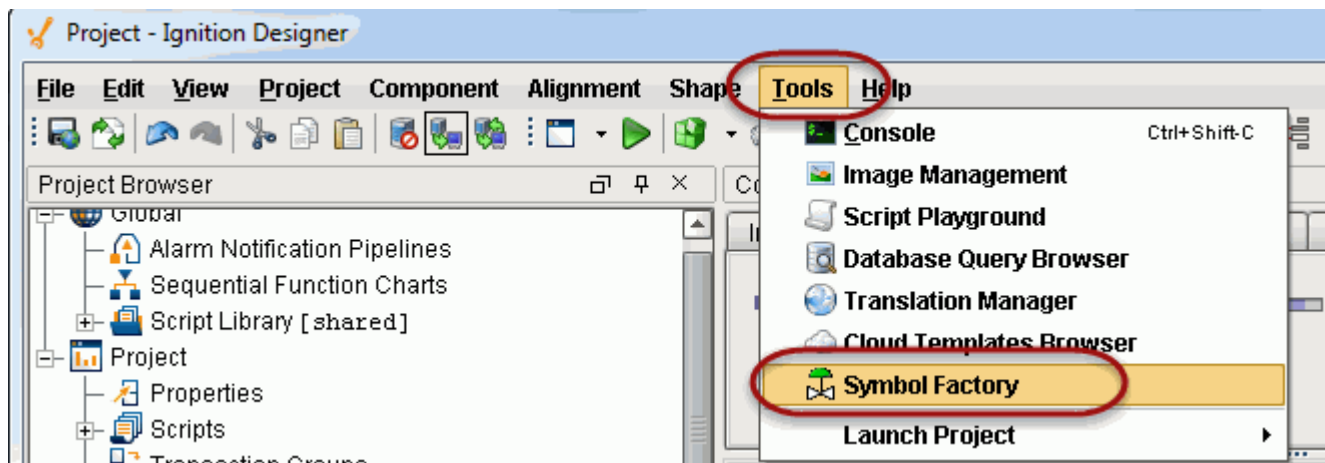
Now that we have a tag for each of the 5 PLCs, let's display the status of each PLC. For example in our **Header** window, we can make a PLC graphic and color it based on the state: **0 = red, 1 = green**.

6. In the **Project Browser**, double-click the **Header** window to open it.

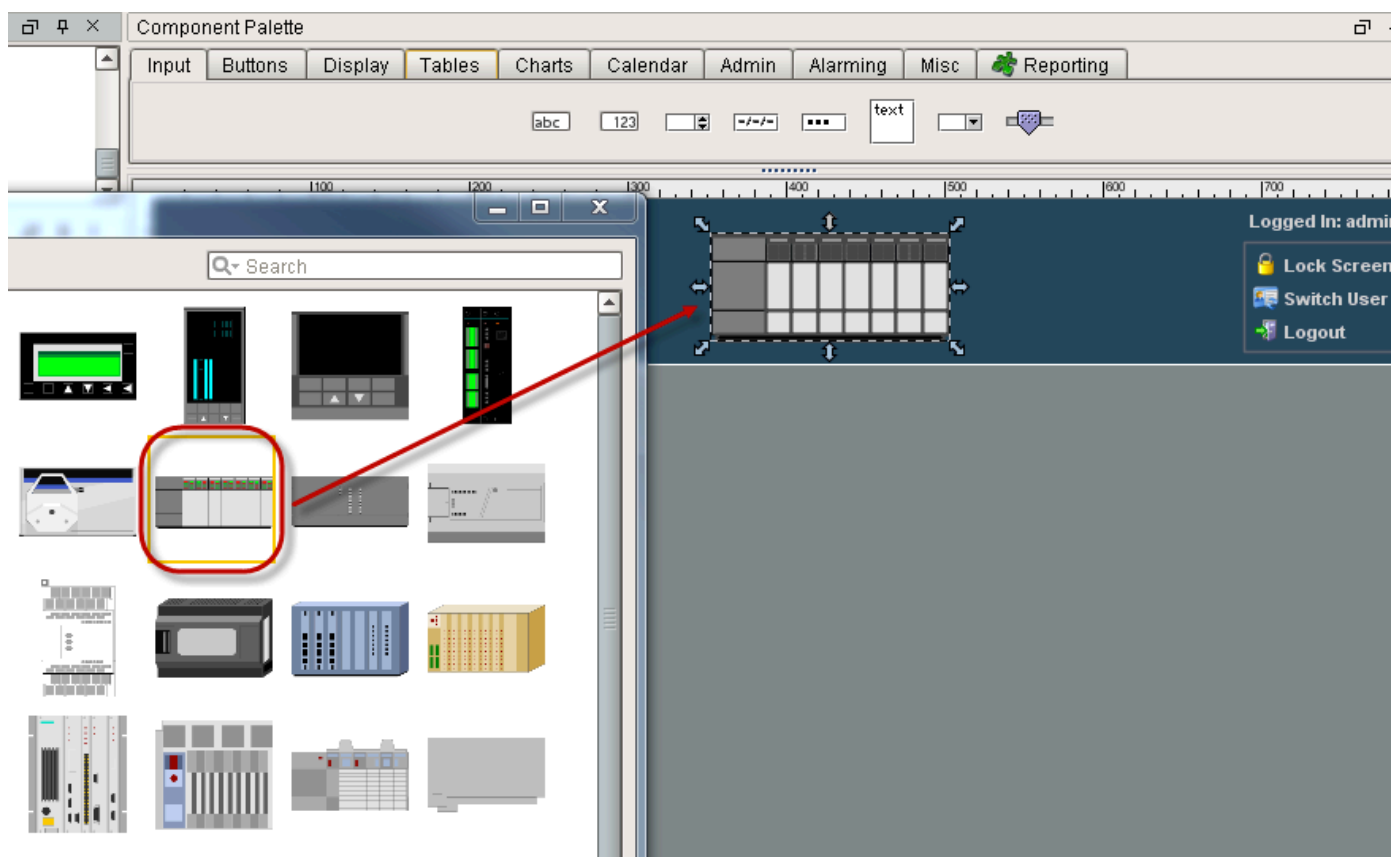


We will add a PLC graphic from Symbol Factory to the Header.

7. From the **Tools** menu, select **Symbol Factory**.

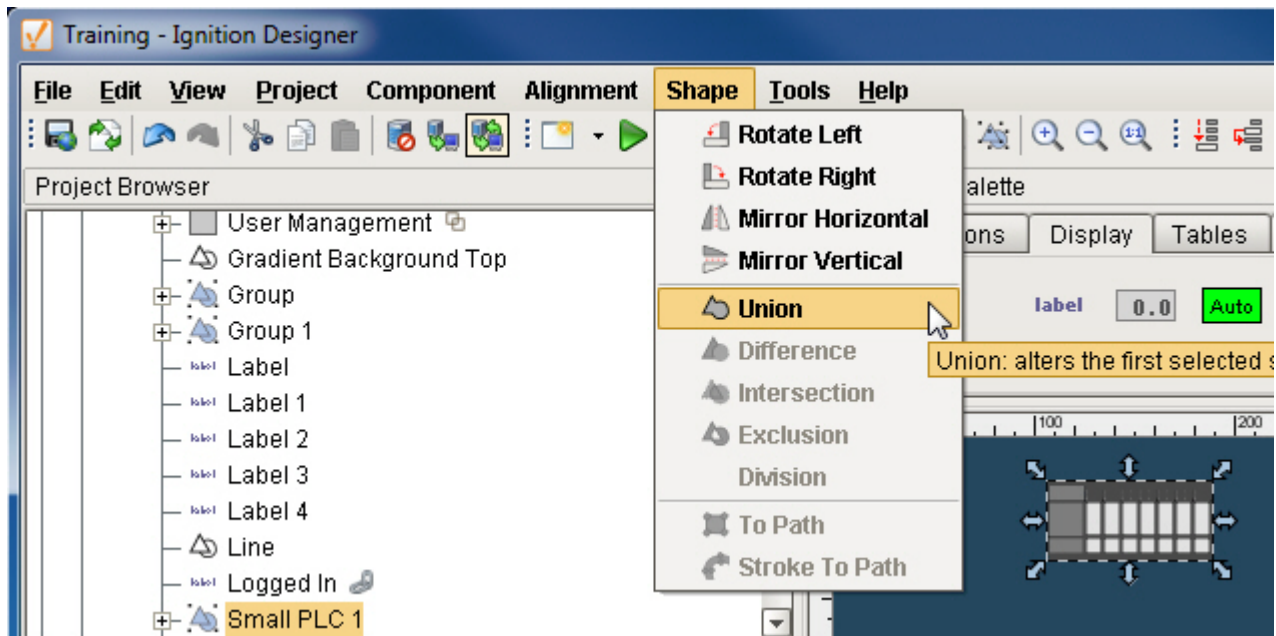


8. From the Symbol Factory window, select **Controllers** in the list.
9. Drag any device on the window to create the component (the PLC graphic).

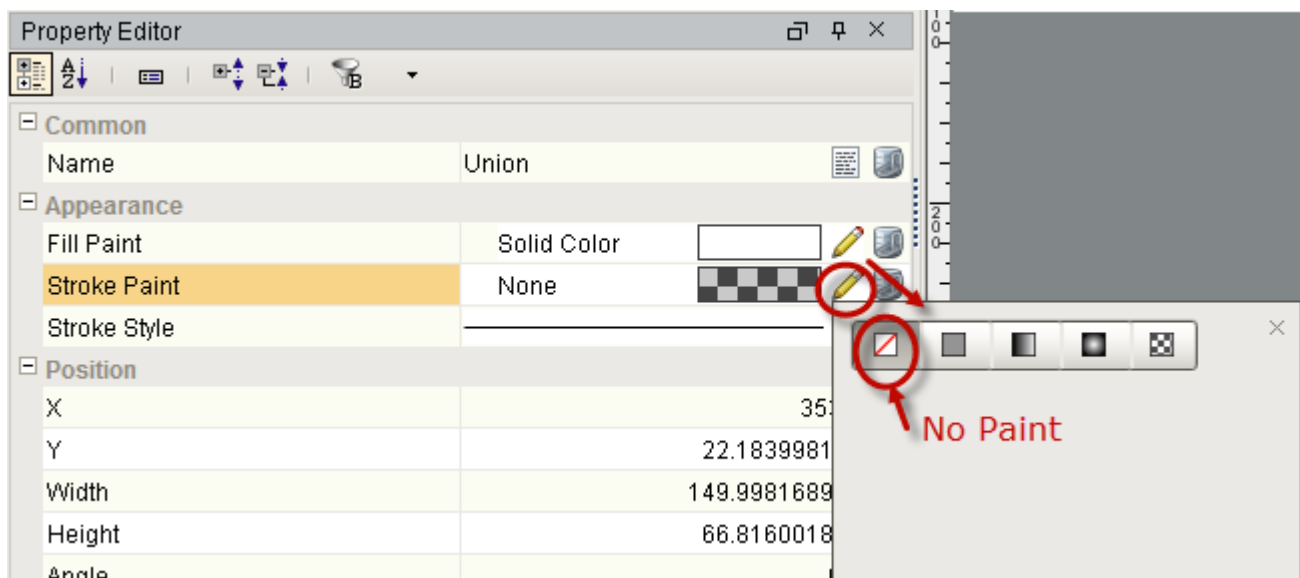


Symbol Factory components are made of several subcomponents or paths that make up a vector-based graphic. If you expand the component in the Project Browser, you can see all the paths. You can also double-click on the component in the design space to select the individual subcomponents. Fill color can be applied to each of these sub-components (or paths) dynamically through property binding. But there are two problems with applying color to the entire component. One is that you would have to change the fill color of each subcomponent dynamically through property binding which can take some time. Two, you will notice that you cannot bind the fill paint to a gradient color. To resolve these two problems we use a special method as described here to apply a tint to the entire component.

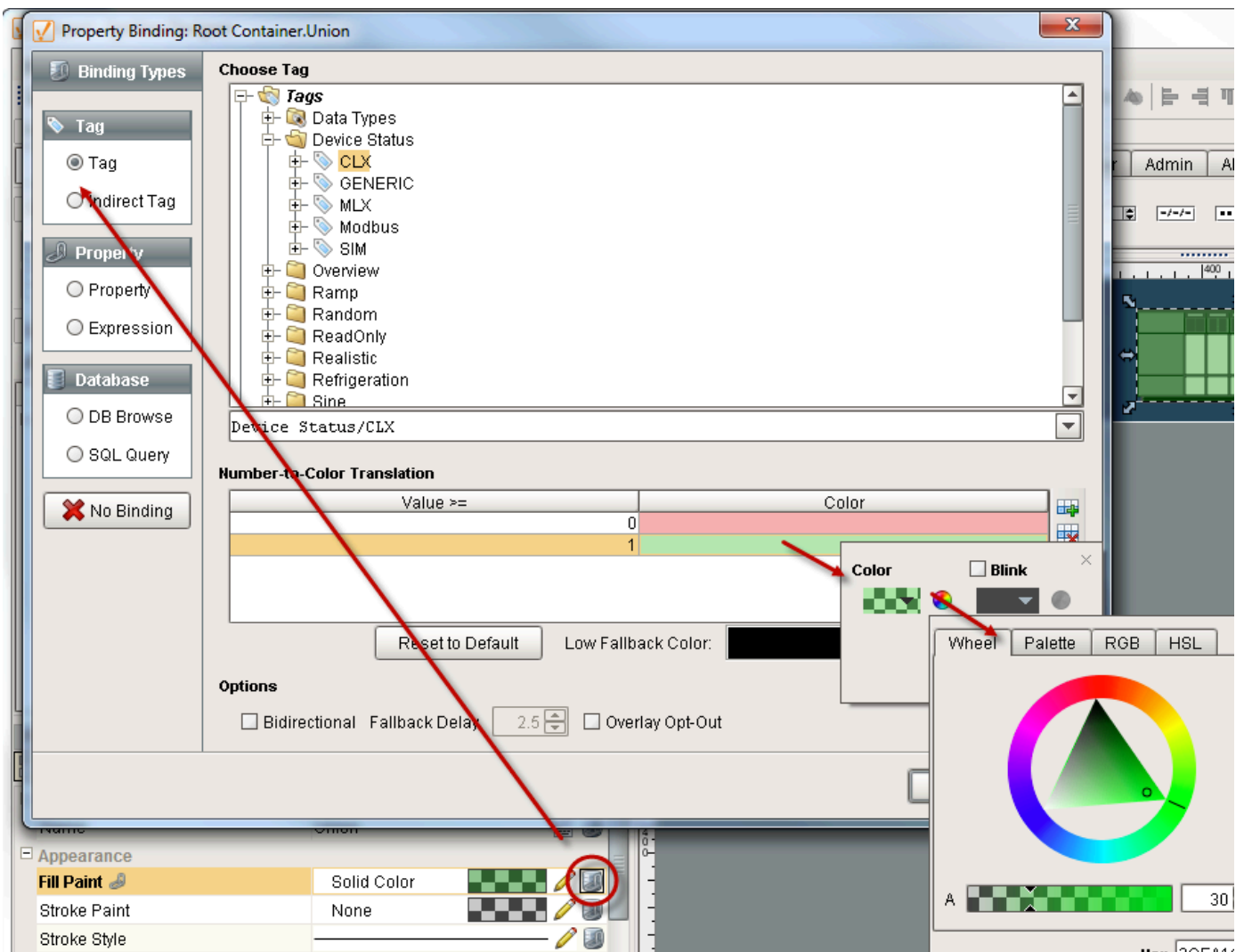
10. To apply tint to the entire component, duplicate the component by selecting it and then select **Edit** > **Duplicate** from menu or press Ctrl-D.
11. Select the duplicated component, which is above the original component.
12. Select **Shape** > **Union** from menu or click the **Union** button in the toolbar. This flattens the duplicated shape into a single shape.



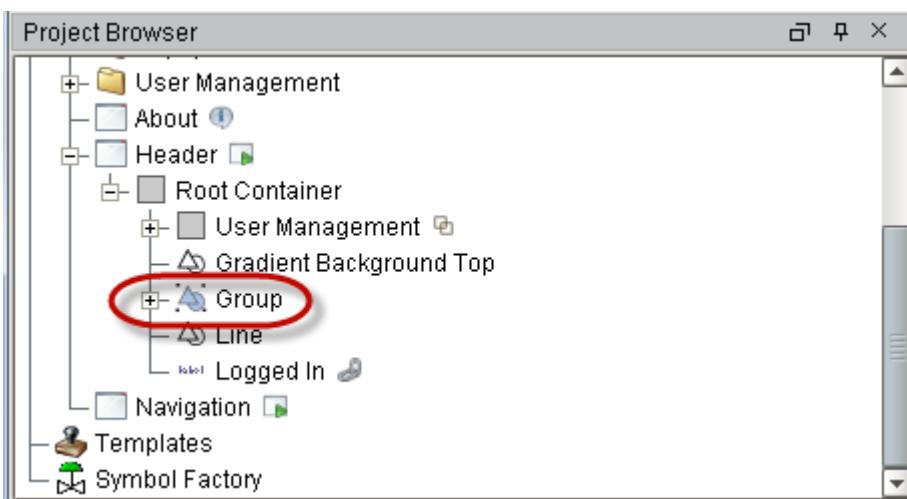
13. Remove the outline of the shape by clicking on the pencil icon of the **Stroke Paint** property and selecting **No Paint**.



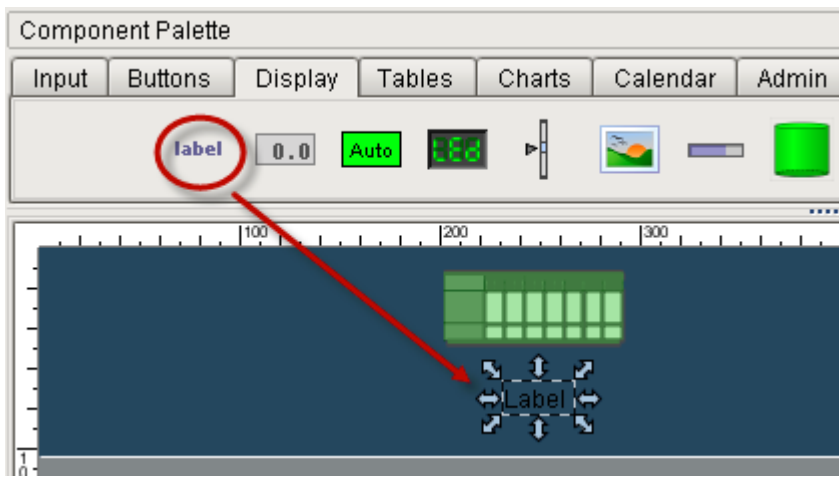
14. Bind the **Fill Paint** property to the **CLX** status tag. Notice when binding a color property to a tag, you get to choose a color for each state of the tag in **Number-to-Color Translation**. You can also make the colors to blink between two different colors.
15. In **Number-to-Color Translation**, set value **0 = red**, value **1 = green**, and alpha channel of each color to **30** (make sure each color is semitransparent so you can see the actual symbol behind it).



16. Group the two components together into one by selecting both components from the Project Browser and clicking the **Component > Group** or Combine icon in the toolbar. Now they are one component.

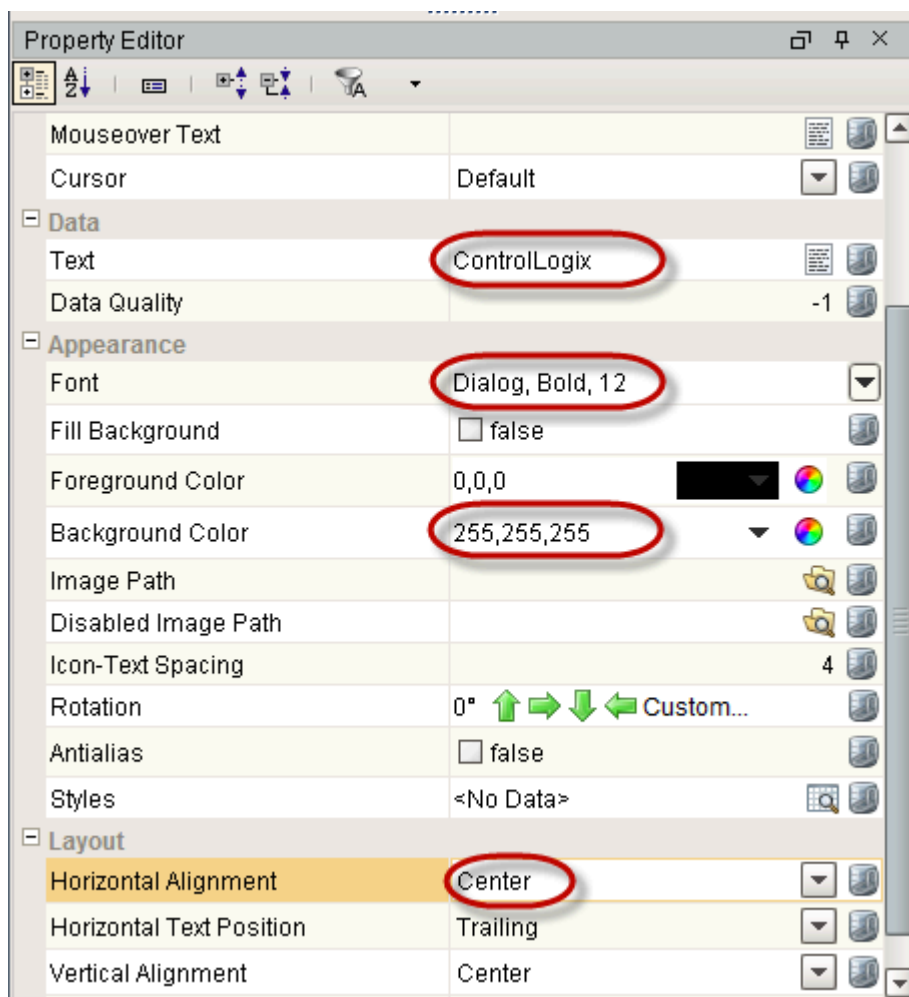


17. To name the device, drag in the **label** component from the **Display** tab into the window under the graphic.



18. Set the label properties as follows:

- Text: **ControlLogix**
- Font: **Dialog, Bold, 12**
- Foreground Color: **255,255,255**
- Horizontal Alignment: **Center**



19. Repeat for each PLC.



You can now see the status of each PLC based on the colors they display (0=red, 1=green).