



Ignition Core Course

5-day instructor-led training



800-266-7798

www.inductiveautomation.com

Ignition

by Inductive Automation

Ignition Core Training

Copyright 2024
All rights reserved.

Inductive Automation
90 Blue Ravine Road
Folsom, CA 95630

5-Day Ignition Instructor-Led Core Training

© 2024 Inductive Automation

All rights reserved. No part of this work may be reproduced in any form or by any means - graphic, electronic, photographic or mechanical; including, but not limited to, photocopying, recording, taping or information storage and retrieval systems - without the written permission of Inductive Automation.

All Inductive Automation product names, including, but not limited to, Ignition by Inductive Automation®, Ignition Edge by Inductive Automation®, Ignition SCADA by Inductive Automation®, Perspective®, as well as the company name, Inductive Automation® and Ignition check mark symbol and the Inductive Automation sprocket/arrow symbol are trademarks of Inductive Automation, either through federal registration or continuous use. Other brand and product names are trademarks of their respective holders.

Every effort has been made to make this book as complete and as accurate as possible, but no warranty of fitness is stated or implied. The information provided is on an "as is" basis without any warranty of any kind. Inductive Automation shall have neither liability nor responsibility to any person or entity with respect to any loss or damages from any acts or omissions arising or based upon the information contained in his book.

Revised September, 2024

Version: 8.1.42

This text was written using Ignition 8.1.42. All screenshots and text reflect the state of Ignition as of that version.

Table of Contents

Chapter 1. Software Introduction	11
What Is Ignition?	12
Ignition Software Stack	14
Ignition's Main Parts	15
The Gateway	16
Projects.....	16
Chapter 2. Installation and Activation	17
Install Ignition on Windows	18
Ignition Editions.....	23
Installing Standard Edition.....	24
Accessing the Gateway	29
Gateway Configuration	30
Gateway Status	32
Licensing and Activating	32
Activating the In-Class License.....	33
Chapter 3. Connecting to PLCs and Databases	37
Connecting to Device Simulators.....	38
Connecting to Physical Devices.....	44
Connecting to a Database.....	47
Chapter 4. Launch Designer and Create a Project	51
Installing the Designer Launcher	52
Adding and Opening a Designer	55
Creating a New Project	59
Chapter 5. Introduction to Designer	63
The Designer Interface	64
Panels	64

Chapter 6. Vision Clients, Windows, and Navigation	71
Vision	72
Exploring Our Project.....	73
Duplicating an Existing Window.....	74
Creating New Windows	75
Window Types.....	76
Adding Windows to Navigation	79
Modifying Components on a Window.....	82
Changing the Header Icon	83
Previewing in the Client.....	84
Downloading the Vision Client Launcher	86
Chapter 7. Vision Component Properties and Bindings	95
Configuring Components	96
Adding Components to Windows	96
Understanding Component Layout	98
Component Properties.....	101
Use the Symbol Factory.....	109
Explore the Vision Drawing Tools.....	112
Chapter 8. Tags	117
Main Benefits of Tags.....	118
Adding Tags.....	119
Binding Tags to Components	123
Memory Tags	129
Derived Tags	131
Additional Tag Types	134
System and Vision Client Tags	134
Tag Groups.....	135
Create New Tag Groups	136
Chapter 9. Complex Tags (UDTs)	143
UDT Features.....	144
Terminology	144
Create a UDT	144

Create Instances of the UDT.....	150
Editing a UDT.....	157
Overrides	160
Nesting UDTs and Inheritance	160
Chapter 10. Templates	161
Create a Template.....	162
Template Parameters	163
Adding Components to Templates	165
Create an Expression With a Template Parameter	168
Indirect Tag Bindings	169
Adding Template Instances to Windows.....	174
Use Parameters on a Symbol Factory Image.....	180
Select Instances From a List	186
Chapter 11. Tag History	189
Log the Data.....	190
Enable Tag History on a UDT	192
Other Tag History Properties.....	193
Viewing the Tag Historian Database Tables.....	194
Display Historian Data	198
Customize the Easy Chart	204
Chapter 12. Transaction Groups	209
Types of Transaction Groups	210
Group Items	211
Triggers.....	211
Creating a Basic History Group	212
Add a Trigger.....	216
Creating a Recipe Group	219
Chapter 13. Security	233
What We Can Secure	234
Authentication Profiles.....	234
Add Users and Roles	235

Component Security.....	240
Adding Security to a Window	250
Project Security Settings	252
Gateway Security	255
Chapter 14. Alarming	259
Alarm Basics.....	260
Alarm Properties	260
Configure an Alarm.....	264
Viewing Alarms	265
Add UDT Alarms	267
Alarm Status Table	269
Filter the Table.....	272
Custom Display Paths.....	273
Alarm Associated Data.....	274
Chapter 15. Alarm History	277
Create an Alarm Journal	278
Viewing the Journal	280
Configure the Alarm Journal Table	282
Alarm Event Ids	286
Alarm Journal Table Features.....	287
Chapter 16. Alarm Notification	289
Alarm Notification Process	290
Add User Contact Information	290
Create a Roster	292
Create an Alarm Notification Profile.....	294
Alarm Notification Pipelines	298
Attach an Alarm to a Pipeline	303
Testing the Pipeline	304
Escalation Pipeline	308
Dropout Condition	318

Chapter 17. Popup Windows	321
Creating a Popup Window.....	322
Opening a Popup	329
Creating an Indirect Easy Chart	334
Creating a Dynamic Alarm Display.....	342
Test the Finished Popup	344
Chapter 18. Scripting	347
What Is Python?.....	348
Getting Started in Python.....	348
Python Syntax	350
Where Is Python Used?	351
Creating and Accessing Variables.....	351
Adding Code to Components.....	354
Component Scripting Dialog.....	355
Basic Math	359
Conditional Processing	360
Lists	364
Loops.....	368
String Formatting.....	368
Python in Ignition	372
Using the Documentation	378
Reading Component Values	380
Component Extension Functions.....	382
Client and Gateway Event Scripts	389
Chapter 19. Perspective	393
Perspective and Web Terms	394
Creating a View.....	394
Positioning and Sizing Components	397
Bindings in Perspective	400
Previewing in the Browser.....	403
Perspective Views as Templates	403
Perspective Expressions	411
Perspective Indirect Tag Bindings	412

Adjusting Layout Properties	415
Color Overlay.....	418
Embedding Views.....	423
Breakpoint Layout.....	429
Styles.....	432
Perspective Security.....	440
Perspective Navigation	445
 Chapter 20. Reporting	 453
Creating a Basic PDF Report.....	454
Adding Data to Your Report	456
Designing the Report	462
Add a Chart	467
Add a Table	470
Report Scheduling	478
Client Reports	481
 Chapter 21. Gateway Network	 485
Connect to a Gateway	486
Set Up a Remote Tag Provider	488
 Chapter 22. Backups	 491
Gateway Backup	492
Project Backup and Restore	495
Backup Tags.....	497

Welcome

Thank you for choosing Ignition by Inductive Automation, a highly configurable industrial automation SCADA/HMI software.

About This Training

We will use a step-by-step approach to teach you the core features and capabilities offered by Ignition. We focus on the Ignition platform and several of its core modules.

Although the architecture of Ignition is modular, the modules are seamlessly integrated so you can focus on the tasks being performed.

Some of the things you will learn in this training are: how to install the software, connect to PLCs and SQL Databases, create projects, create tags and store history, launch clients and sessions, set alarms, send notifications, and generate reports.

Let's get started!

Software Introduction

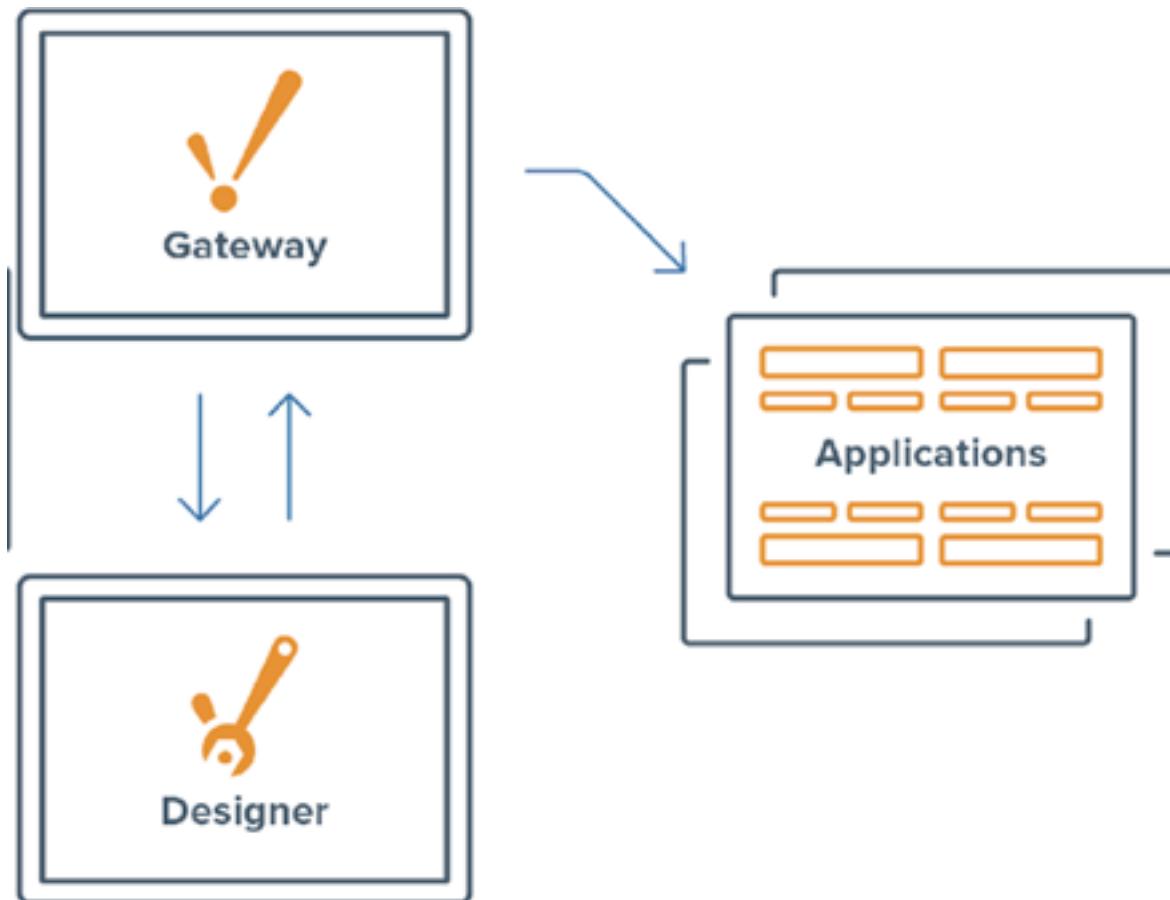
This chapter introduces you to Ignition. It describes what Ignition is, what it can do, what modules are, and provides an overview of common Ignition system architectures.

What Is Ignition?

Ignition is a software platform for creating custom human machine interface (HMI), supervisory control and data acquisition (SCADA), manufacturing execution system (MES), and Industrial Internet of Things (IIoT) applications. Ignition is designed from the ground up to be approachable and easy to get started with while remaining flexible and capable of scaling up to the largest projects. Ignition is installed as server software and it is based heavily on current web technologies.

Ignition's Basic Flow

Ignition is unique in the industrial space because it is server-centric. You launch a Designer and pull data from the Gateway. Then when you save your project, it's stored back into the Gateway. When runtime clients are opened, they pull from the Gateway as well. This means any client that has a connection to the Gateway will always be up to date.



What Can Ignition Do?

Ignition has a unified architecture that provides a way to create anything you need. Whether you need a custom SCADA application, or need to visualize historian data, or want to generate PDF reports, or pure database applications, or work notify operators of alarms, or more, Ignition is the right solution. You can also mix and match to get the functionality that you want and leave out the rest.

What Makes Ignition Special?

There are a few key points that set Ignition apart from other HMI/SCADA software. The licensing is sold by the server and allows unlimited tags, clients, screens, connections, and devices. It has a small installer that downloads and installs in minutes across Windows, Mac, and Linux. The Designer is included and allows you to quickly create projects, bind values, set up expressions, and add scripts. It's modular so it's flexible and scalable. And it's easy to get started and see real results in just a few minutes.



Hassle-Free Licensing



Easy Installation & Deployment



Rapid Design & Development



Flexible & Scalable



Easy to Get Going

What Are Modules?

Modules are applications that are built on the Ignition platform. Modules are seamlessly integrated into Ignition so you can choose them based on your system and plant requirements. A few common modules include OPC UA connections and drivers, visualization systems, history storage, and alarm notification.

Ignition Software Stack

The architecture or high-level structure of the Ignition software has several distinct layers. The platform runs in Java, so as such it is OS agnostic. The modules are in the platform and can access any functionality. Additional modules are built to work using the core modules' functionality as a starting point.



Ignition As a Communications Hub

You can use Ignition as an effective communication hub in your network. The Gateway can talk to many types of devices and services, even if those individual elements are incompatible with one another.



Ignition's Various Architectures

Ignition combines web servers, databases, and OPC UA. This combination allows a wide variety of powerful configurations based on the customer's system and plant needs. There are several types of architectures that can be built with Ignition starting from a standalone installation and going all the way to a multi-site system spread through different countries. Ignition can be used in the cloud if you want, and has fail over redundancy as a primary feature.

Ignition's Main Parts

Ignition has four main parts: Gateway, Designer, Clients, and Sessions.

Gateway

The primary service that drives everything. It is a single application that runs an embedded web server, connects to data, executes modules, communicates with clients and more.

Designer

A web-launched application that lets you configure and build your projects. The Designer is launched from the Designer Launcher that can be found on the Gateway.

Clients

The runtimes of the Vision Module. They run as full applications and feel like traditional installed clients without the need to install and manually synchronize projects. Clients are launched from the Designer Launcher that can be found on the Gateway web page.

Sessions

The runtimes of the Perspective Module. They are similar to the Vision Clients except they run in a browser. Sessions are launched from the Gateway web page.

The Gateway

The Gateway's architecture contains many parts, each one performing specific tasks or serving up information to other systems. The Gateway contains database and device connections, tags, alarms, user sources, projects, and more.

Projects

A project is the central unit of configuration. A project contains Windows, Views, Transaction Groups, and more. You use the Ignition Designer to design and create projects. There is no limit to the number of projects you can create in the Gateway.

Installation and Activation

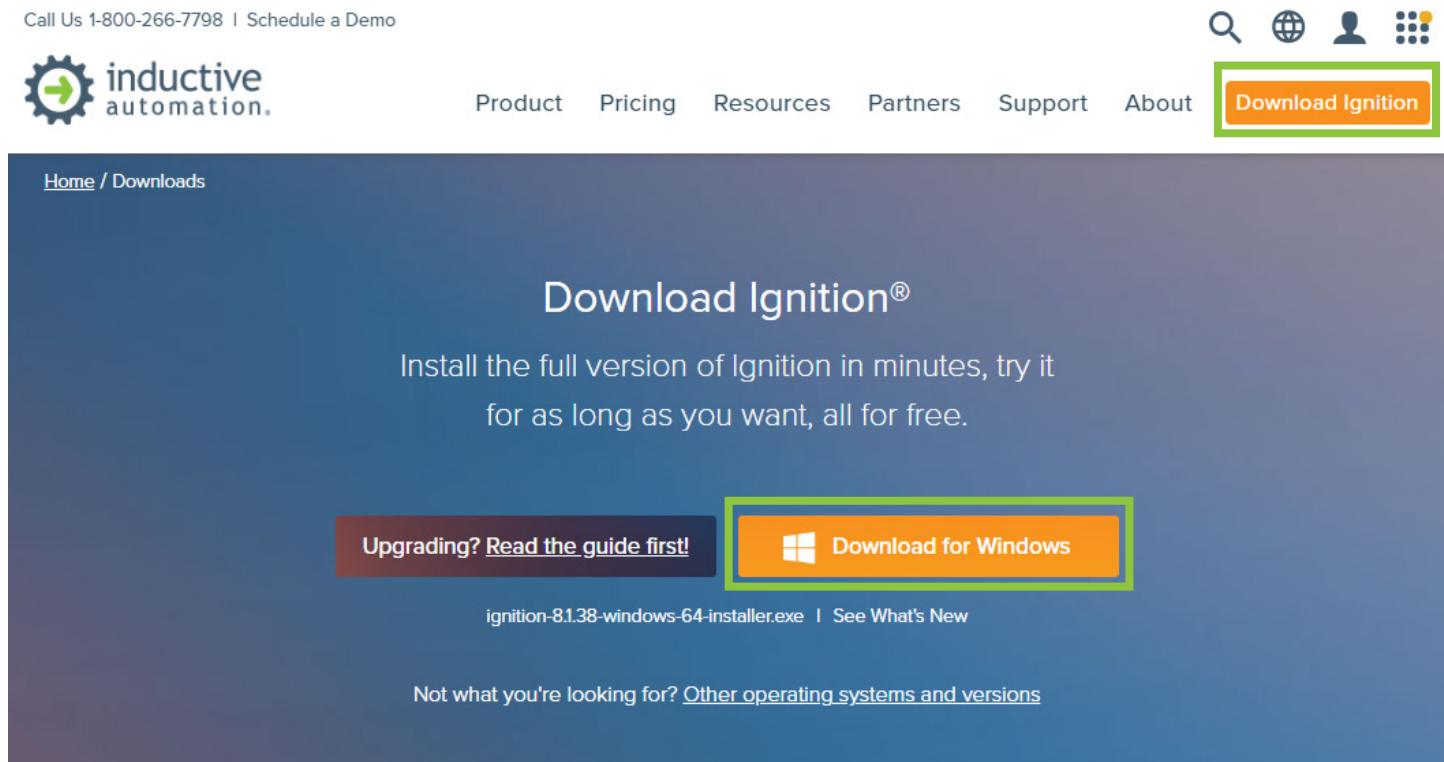
Ignition can be installed on any 64-bit Windows, Linux, or Mac OS X operating system. The installer can be downloaded from the Inductive Automation website and installed on any modern computer for testing. When using Ignition in production, a server-class computer is recommended.

The minimum system requirements for Ignition are extremely low, but we recommend at least a dual-core processor, 4GB RAM, and 10GB free hard drive space.

Install Ignition on Windows

Installing Ignition on your computer is just like installing other applications.

1. Visit [ia.io](#) on any modern web browser.
2. Click the orange **Download Ignition** button in the top right corner.



3. Click **Skip form and download directly**.

- 4.

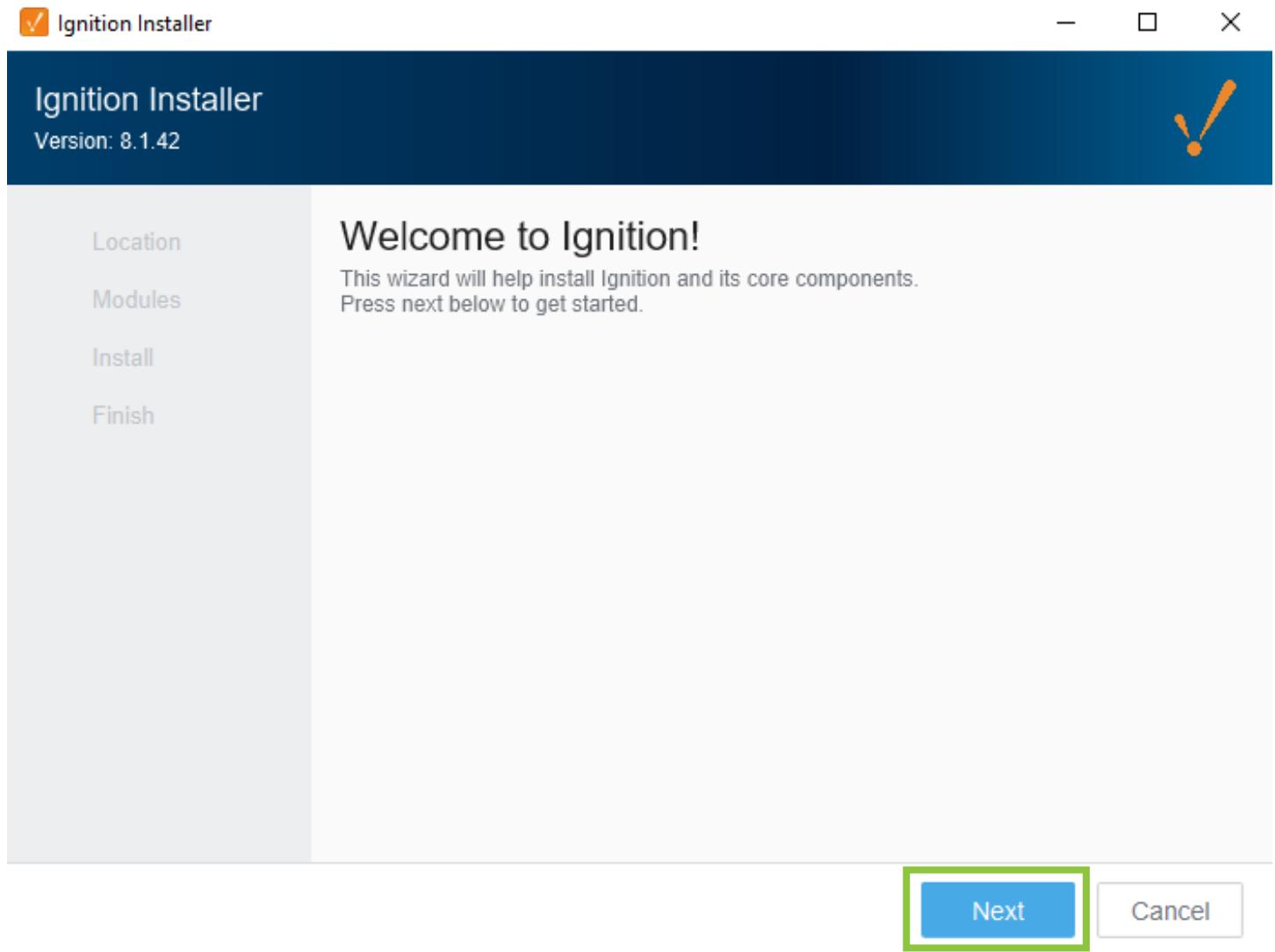
By clicking "Download," you agree to the [Terms of Use](#), and the submission and processing of your data. Your privacy is very important to us. [Privacy Policy](#).

[Skip form and download directly >](#)

[Download](#)

5. Open **Windows Explorer**.

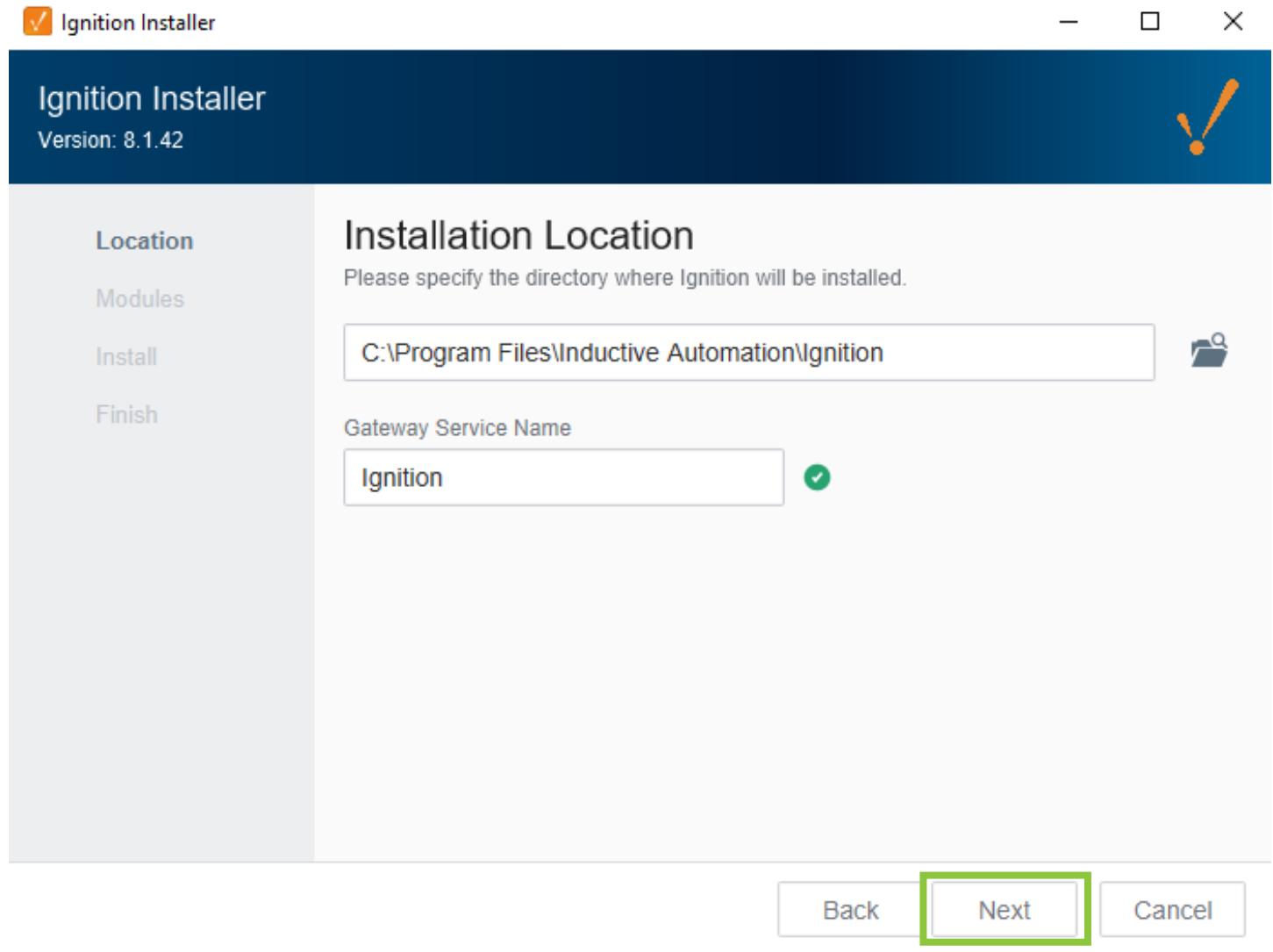
6. Find the installer file. Most modern browsers will have automatically saved it to the Downloads folder.
7. Double-click on the **executable file** to start installation.



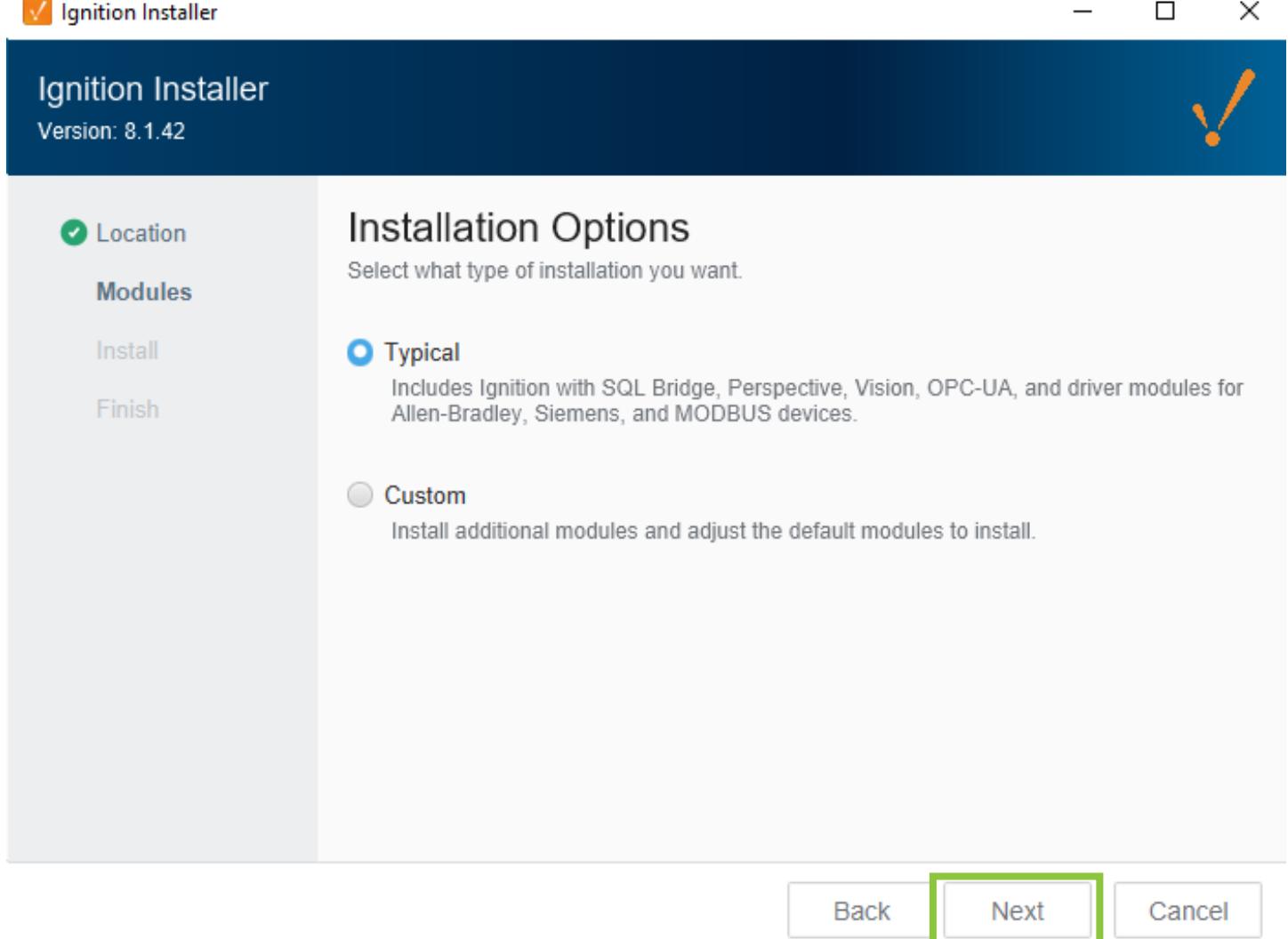
The **Ignition Installer** starts.

8. Click **Next**.

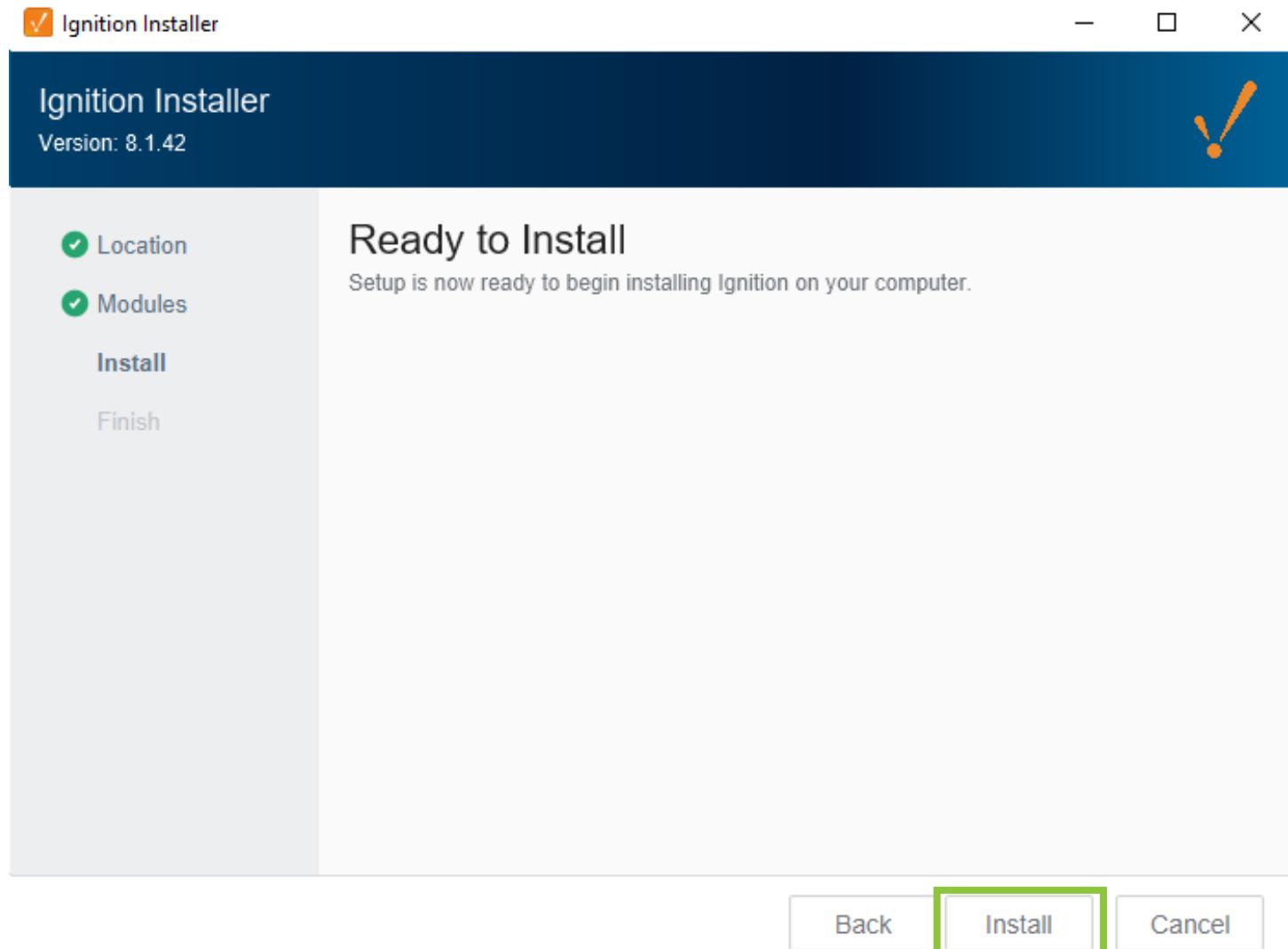
9. Click **Next** to accept the default installation file location and Gateway Service Name.



10. Click **Next** to select the Typical installation.

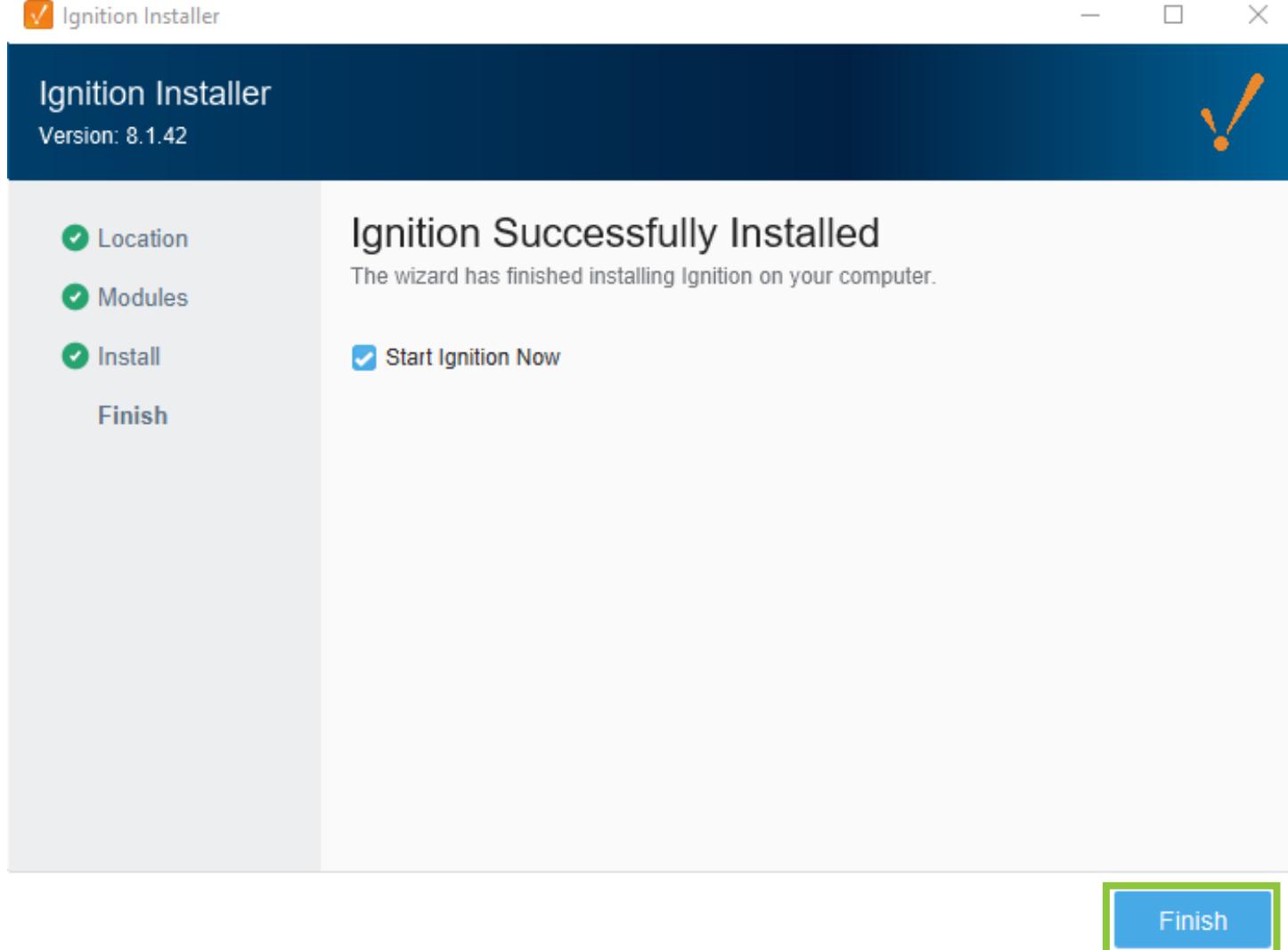


11. Click **Install**.



Installation begins.

12. Click **Finish**.



The Ignition service will start. After a few moments, the Gateway web page will open.

Ignition Editions

You can choose to install Ignition using one of three editions.

Standard Edition

The Standard edition, which we will be using in this class, is the most common edition for Ignition installations, and should be used in all situations except those outlined below.

Maker Edition

The Maker edition is a free, limited edition of Ignition for personal, non-commercial use. There are a limited number of modules available for Maker edition. You can get more information on Maker on our website at <https://inductiveautomation.com/ignition/maker-edition>.

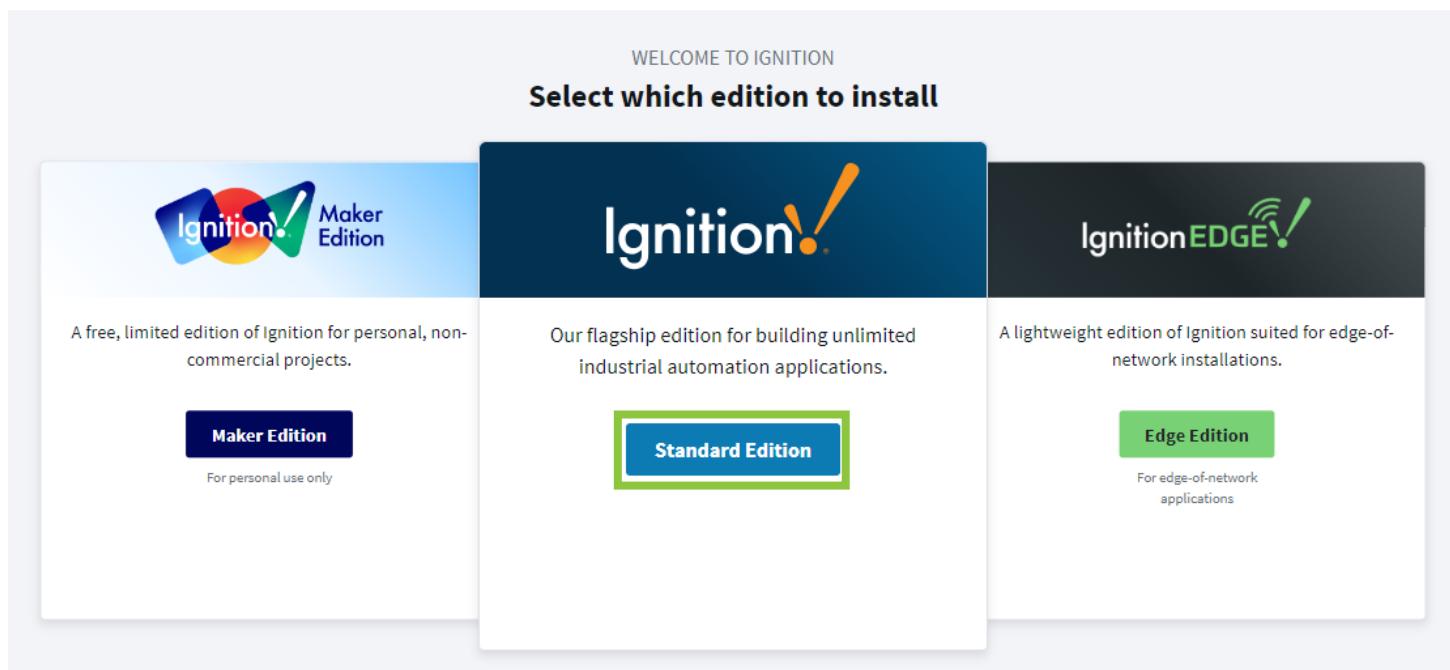
Edge Edition

Ignition Edge is designed for edge-of-network installations. More information on Edge is available on our website at <https://page.inductiveautomation.com/ignition-edge>.

Installing Standard Edition

To finish installing Ignition, follow these steps.

1. Click **Standard Edition**.



2. Click **I have read and agree with the terms and conditions.**

3. Click **Next.**



Inductive Automation Software License Agreement

IMPORTANT - Read this License Agreement carefully before clicking the Agree button. By clicking on the Agree button, you agree to be bound by the terms of the License Agreement.

INDUCTIVE AUTOMATION END USER LICENSE AGREEMENT

Last updated: February 16, 2023

IMPORTANT - READ THIS AGREEMENT CAREFULLY:

**THIS END USER LICENSE AGREEMENT (“EULA”) IS A LEGAL AGREEMENT
BETWEEN YOU (EITHER AN INDIVIDUAL OR A SINGLE ENTITY) AND INDUCTIVE
AUTOMATION, LLC. BY CLICKING ACCEPT, INDICATING ACCEPTANCE,**

I do not agree with the terms and conditions. [Open in new tab](#)

I have read and agree with the terms and conditions.

• • • Step 1 of 3 **Next →**

4. Create a **user**.
5. Enter a username of **admin**.
6. Enter a password of **password**.
7. Confirm the password.

For class purposes, we want to ensure that we all have the same username and password, even though these credentials should never be used in a real environment.

8. Click **Next**.

The screenshot shows the 'Create a User' interface. At the top, the Ignition logo is displayed. Below it, the title 'Create a User' is shown. A descriptive text explains that the user will have full administrative privileges by default, which can be edited later in the Gateway. The form contains three input fields: 'Username' (containing 'admin'), 'Enter Password' (containing 'password'), and 'Confirm Password' (containing 'password'). To the right of the 'Enter Password' field is a 'Password Strength' meter, which is red and labeled 'Very Poor' with a score of 25. A note below the meter states: 'Your password is easily guessable. You can do better.' At the bottom left, there are two blue dots indicating this is Step 2 of 3. At the bottom right, a large blue 'Next →' button is visible.

Ignition!

Create a User

Take a moment to create your first user account. This user, by default, will have access to full Administrative privileges in Ignition. This can all be edited later in the Gateway.

Username

admin

Must start with a letter or digit and contain only letters, digits, spaces, underscores, @, periods or dashes. Must be 2-50 characters.

Enter Password

.....

Password Strength

Very Poor 25

Your password is easily guessable. You can do better.

Confirm Password

.....

• •

Step 2 of 3

Next →

Ignition requires three ports in order to run. The HTTP port, which defaults to 8088, is used for Ignition's built-in web server and is how we will access the Gateway web

page. The HTTPS port, which defaults to 8043, is used when you access the Gateway web page in a secure environment and need to have the communication between the Gateway and the end-user browser encrypted. The Gateway Network Port, with a default of 8060, is used in internal network communications.

You should discuss with your IT department if these ports need to be changed in a production installation of Ignition. For testing and class purposes, the defaults work well.

9. Click **Finish Setup**.

The screenshot shows the Ignition setup interface. At the top, there is a large Ignition logo. Below it, the title "Configure Ports" is displayed. A descriptive text follows, stating: "Configure which ports you would like the Ignition Gateway to bind to. If you're unsure, leave the defaults, they work well in the majority of situations." Three input fields are present for port configuration:

- HTTP Port:** The value is set to 8088, with "(default: 8088)" noted below it.
- HTTPS Port:** The value is set to 8043, with "(default: 8043)" noted below it.
- Gateway Network Port:** The value is set to 8060, with "(default: 8060)" noted below it.

At the bottom left, there is a progress indicator showing three dots, with "Step 3 of 3" next to it. On the bottom right, there is a blue button labeled "Finish Setup →".

10. Click **Start Gateway**.

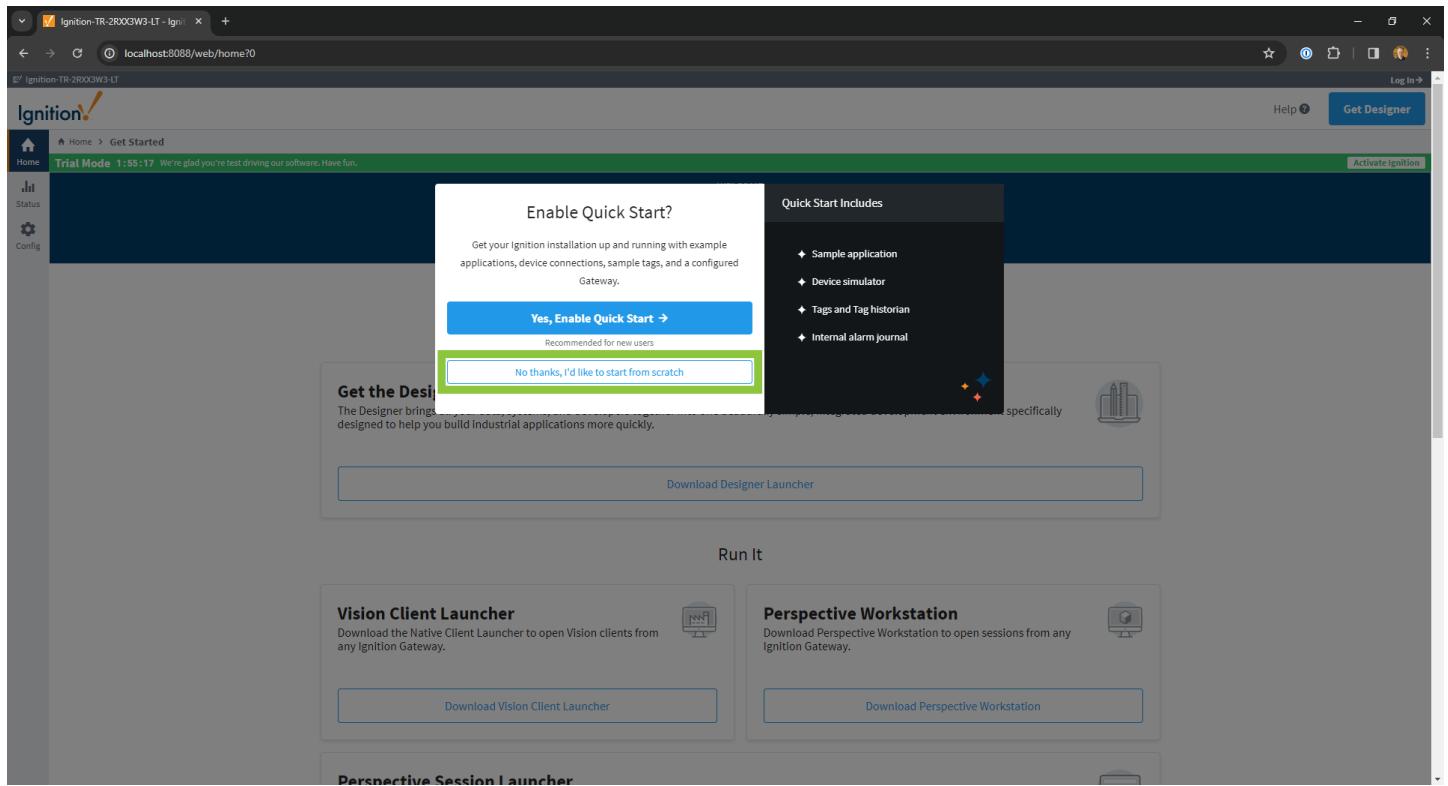
SETUP COMPLETED

Start and Launch the Gateway Now?

Start Gateway

11. Click **No thanks, I'd like to start from scratch.**

We do not need any of the resources created by Quick Start.



Accessing the Gateway

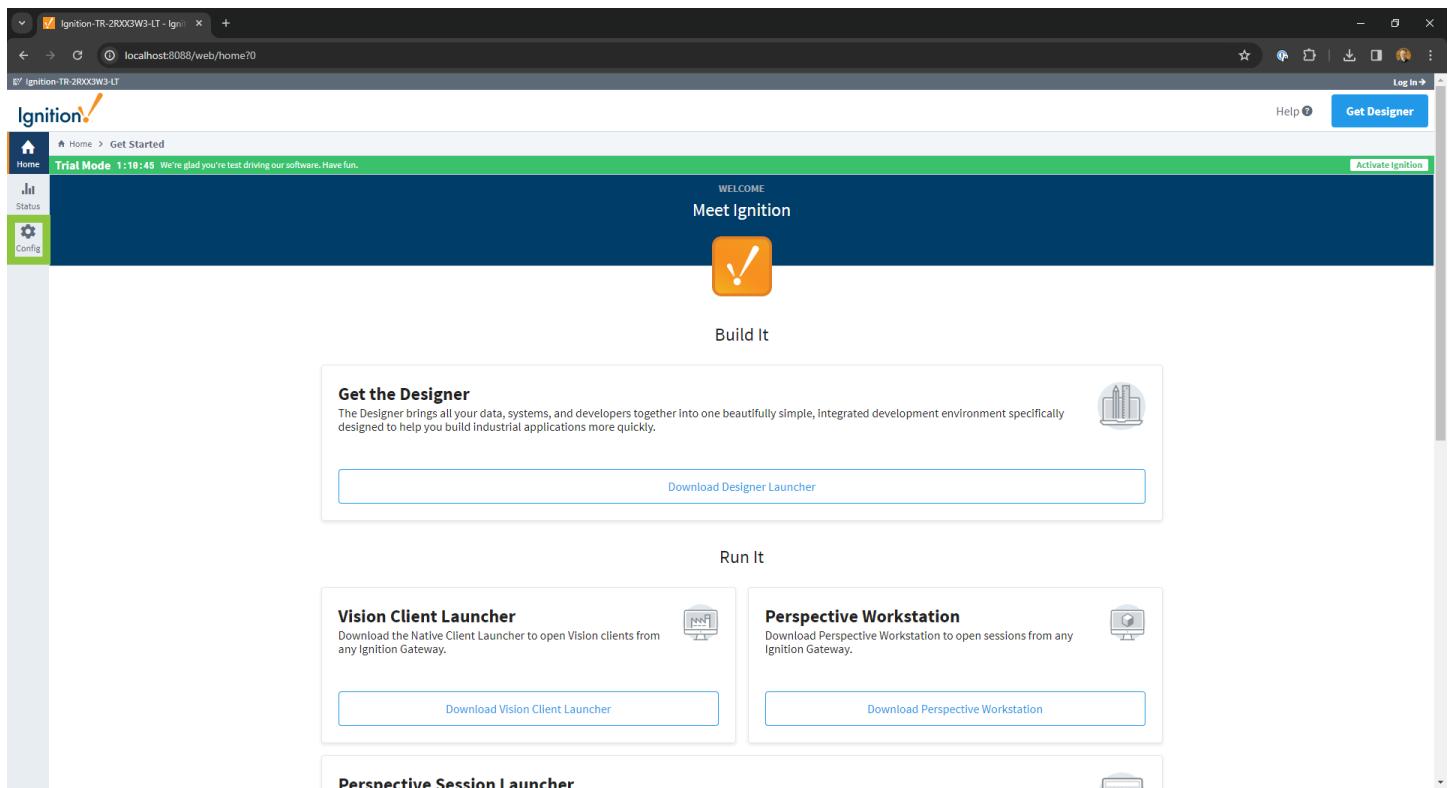
The Ignition Gateway is accessed through a web browser and is the primary software service that drives everything in Ignition. It is a single application that runs an embedded web server, connects to data, executes modules, communicates with clients and sessions, and more.

Once we complete setup, we will be on the Gateway web page, but should you ever need to access it again, you can do so by typing <http://localhost:8088> in your browser's address bar. You might also consider saving the web page as a bookmark in the browser.

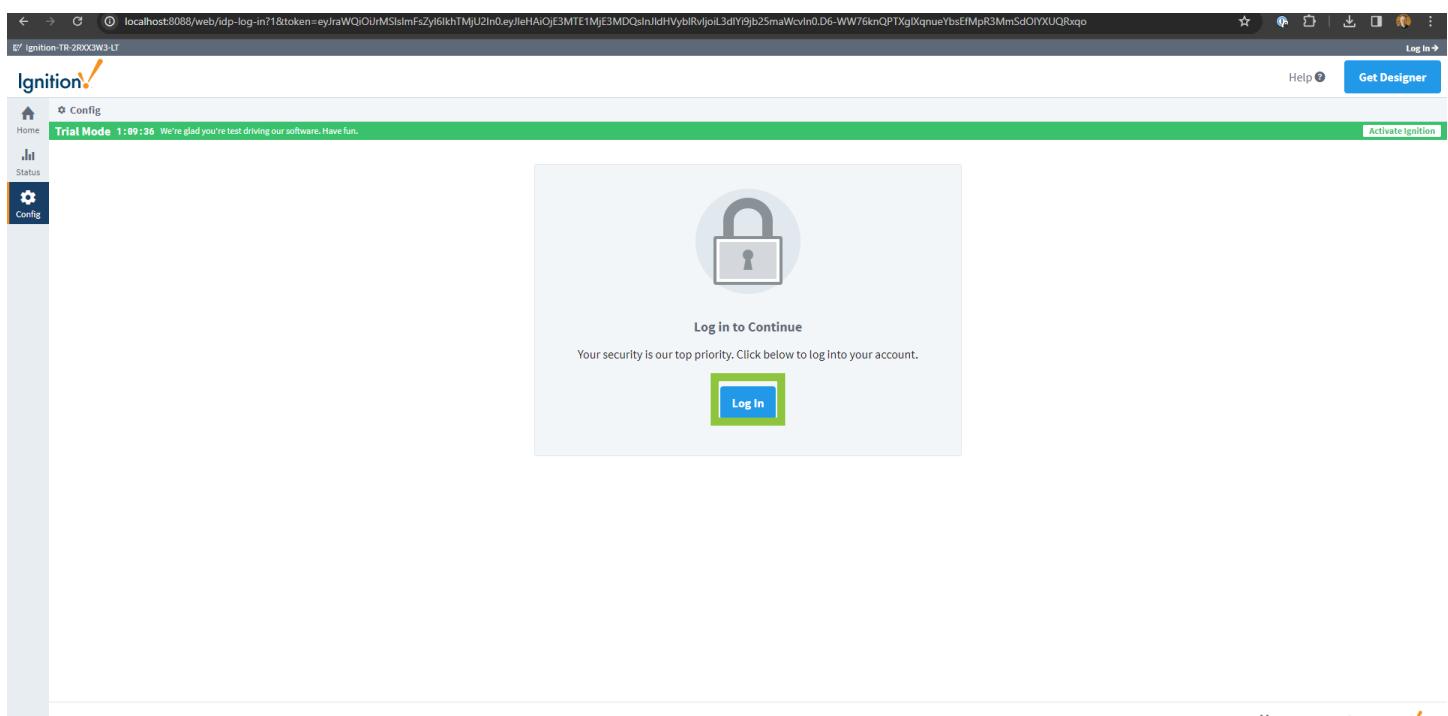
Gateway Configuration

We use the Gateway web page to configure settings within our Ignition installation. To access the configuration settings, follow these steps.

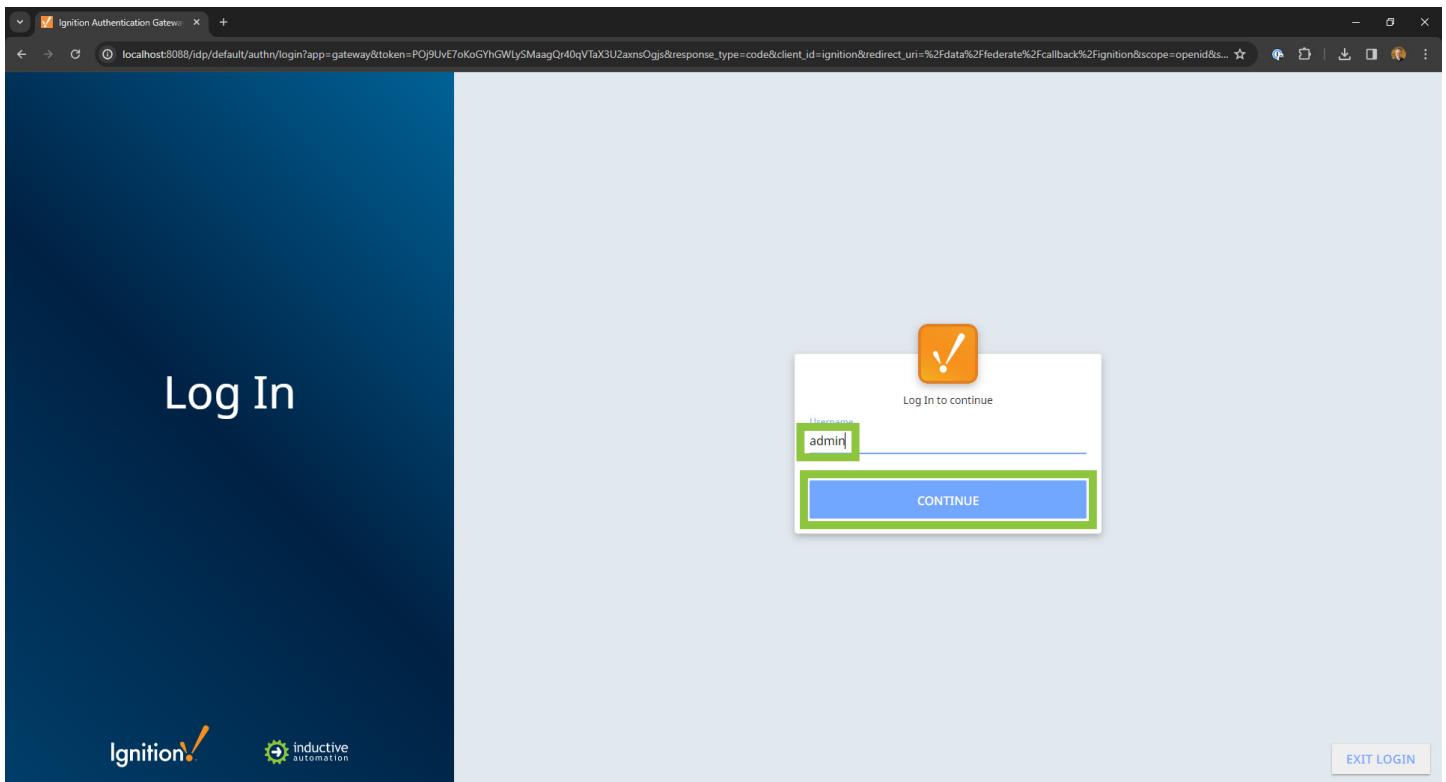
1. Click **Config**.



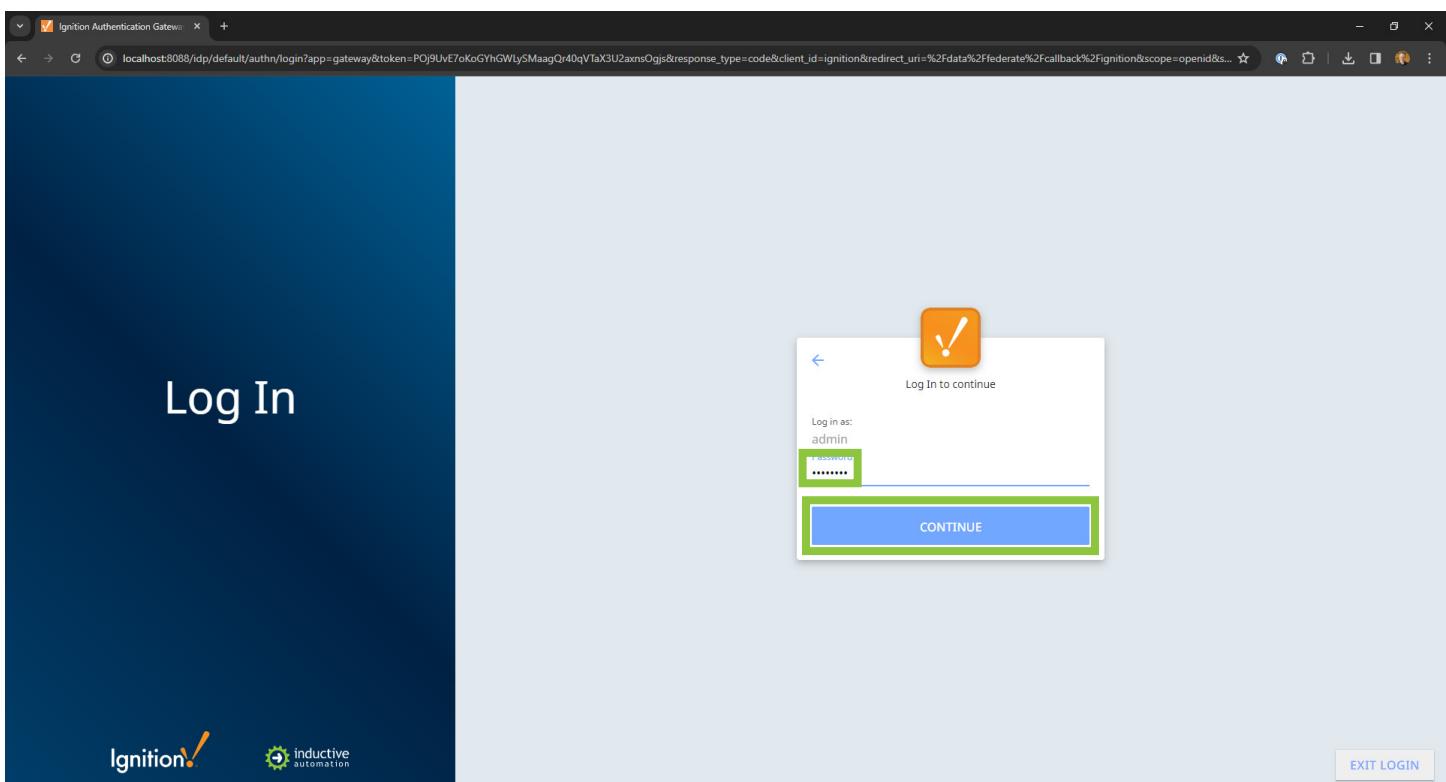
2. Click **Log In**.



3. Enter **admin**.
4. Click **Continue**.



5. Enter **password**.
6. Click **Continue**.



You are now logged into the Configuration section. Here, you can set up connections to devices and databases, configure security and alarming, create backups, and much more.

Gateway Status

The other primary resource provided on the Gateway web page is the status section, which can be accessed by clicking the Status button near the top left corner.

From the Status overview page, you can see what version of Ignition you are running, get information about the operating system and hardware Ignition is running on, and see the status of various subsystems, such as database connections, devices, and more.

Note that the Status section is secured by the same log in credentials as Config. Both will timeout occasionally, so you may need to log back in to access either site.

Licensing and Activating

Ignition has a free trial mode that works for two hours at a time and can be reset an unlimited number of times. For this class, we will activate a week-long license so that we do not have to keep resetting the server.

With this Ignition server license, we will be able to:

- Launch unlimited runtime clients.
- Create unlimited Tags from devices, OPC servers, and other Tags.
- Connect to unlimited PLCs, SQL databases, and devices.
- And much more.

How Licensing Works

Like Ignition as a whole, licensing is based on modules.

Licensed and unlicensed modules can operate side-by-side, so some modules can be tested in trial mode while others are running in a licensed state.

When an individual module is in trial state, Ignition shows the trial time remaining at the top of the Gateway web page.

Despite the modular licensing, there is generally one CD-Key for each server. That is, a single license dictates which modules are currently activated.

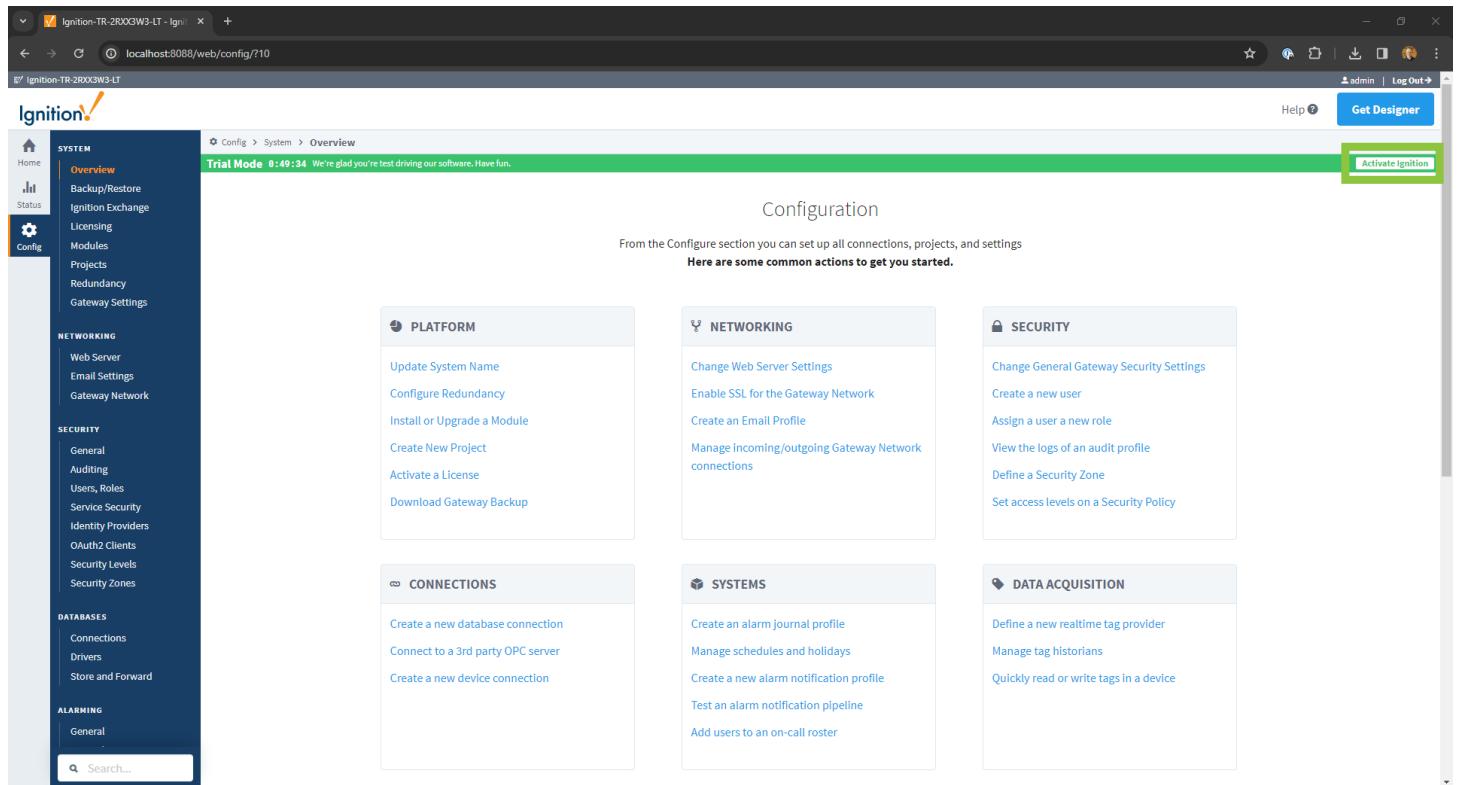
When one or modules are purchased, you will receive a CD-Key, which is a six digit code, that identifies your purchase. You will use this CD-Key to activate the software through the Ignition Gateway. If you are adding an additional module, your account will be automatically updated and you can re-use your existing CD-Key to activate the new features.

You can deactivate your CD-Key at any time, freeing it for activation on a different machine.

Activating the In-Class License

To activate the license for the class, follow these steps.

1. If necessary, **log into the Gateway config site**. See the steps above under “Gateway Configuration” on page 30 for help if needed.
2. Click **Activate Ignition**.



The screenshot shows the Ignition Gateway configuration interface. The left sidebar has sections for SYSTEM (Overview, Backup/Restore, Ignition Exchange, Licensing, Modules, Projects, Redundancy, Gateway Settings), NETWORKING (Web Server, Email Settings, Gateway Network), SECURITY (General, Auditing, Users, Roles, Service Security, Identity Providers, OAuth2 Clients, Security Levels, Security Zones), DATABASES (Connections, Drivers, Store and Forward), and ALARMING (General). A search bar is at the bottom. The main area shows a green banner: "Trial Mode 0:49:34. We're glad you're test driving our software. Have fun." Below it is a "Configuration" section with a sub-section "From the Configure section you can set up all connections, projects, and settings Here are some common actions to get you started." There are six cards: PLATFORM (Update System Name, Configure Redundancy, Install or Upgrade a Module, Create New Project, Activate a License, Download Gateway Backup), NETWORKING (Change Web Server Settings, Enable SSL for the Gateway Network, Create an Email Profile, Manage incoming/outgoing Gateway Network connections), SECURITY (Change General Gateway Security Settings, Create a new user, Assign a user a new role, View the logs of an audit profile, Define a Security Zone, Set access levels on a Security Policy), CONNECTIONS (Create a new database connection, Connect to a 3rd party OPC server, Create a new device connection), SYSTEMS (Create an alarm journal profile, Manage schedules and holidays, Create a new alarm notification profile, Test an alarm notification pipeline, Add users to an on-call roster), and DATA ACQUISITION (Define a new realtime tag provider, Manage tag historians, Quickly read or write tags in a device). A blue "Activate Ignition" button is located in the top right corner of the main content area.

Ch 2. Installation and Activation

3. Click **Activate License**.

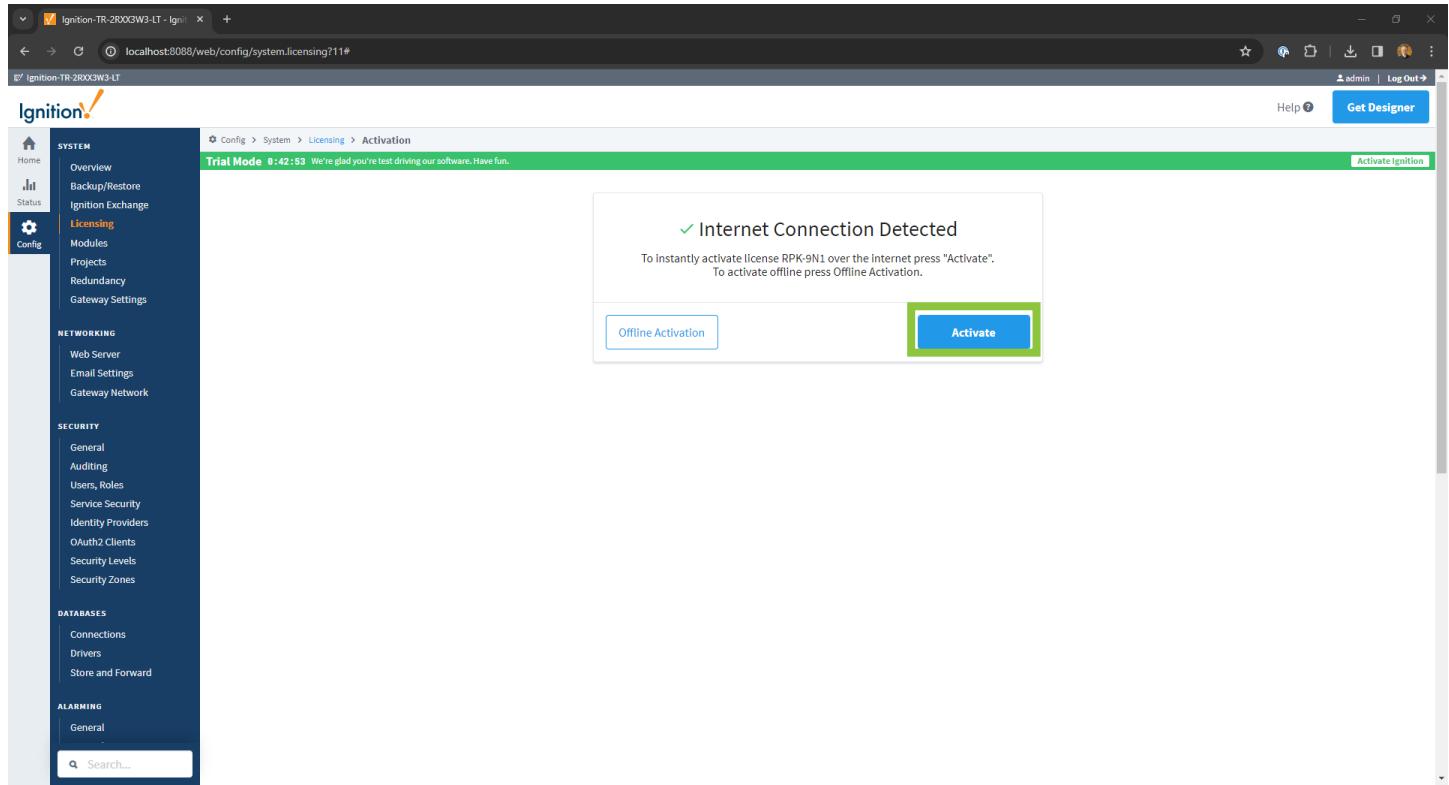
The screenshot shows the Ignition software interface with the title bar "Ignition-TR-2XXXW3-LT" and the URL "localhost:8088/web/config/system.licensing?11". The top navigation bar includes "Help", "Get Designer", "admin | Log Out", and "Activate Ignition". The left sidebar has sections for SYSTEM (Overview, Backup/Restore, Ignition Exchange, **Licensing**, Modules, Projects, Redundancy, Gateway Settings), NETWORKING (Web Server, Email Settings, Gateway Network), SECURITY (General, Auditing, Users, Roles, Service Security, Identity Providers, OAuth2 Clients, Security Levels, Security Zones), DATABASES (Connections, Drivers, Store and Forward), and ALARMING (General). A search bar at the bottom left says "Search...". The main content area displays a large key icon and the message "No Licenses Activated". It explains that trial mode can be used as long as desired, and purchasing a license key will enable activation. Two buttons are present: "Learn More" and "Activate License".

4. Enter the **license key** provided by your instructor.

5. Click **Submit**.

The screenshot shows the Ignition software interface with the title bar "Ignition-TR-2XXXW3-LT" and the URL "localhost:8088/web/config/system.licensing?11#Activation". The top navigation bar includes "Help", "Get Designer", "admin | Log Out", and "Activate Ignition". The left sidebar is identical to the previous screenshot. The main content area is titled "Activate a License" and contains the message "Already have a license key? You're in the right place. Enter your license key below to activate.". It features a text input field and a "Submit" button. Below the input field is a note: "Note: Your license key will be 6 or 8 characters". A callout box provides help for finding the license key: "Need help finding your license key? Check your confirmation email or invoice. If you don't have a license or need help call Inductive Automation at 1-800-266-7798."

6. Click **Activate**.



Your Ignition installation is now licensed.

Once activated, the current license is displayed. You can expand it to see all of the modules attached to your license.

Connecting to PLCs and Databases

Now that you installed Ignition and logged into the Config section of the Gateway, you can add or connect to devices and databases. A device is a named connection to an industrial device like a PLC. If you don't have a real device, Ignition has simulator devices you can add which act as if it is a connection to a real device.

Connecting to Device Simulators

To begin our exploration of devices, we are going to use a programmable device simulator, which will allow us to mimic the functionality of a real PLC.

Connecting to a Simulator Device

We need to connect to a few devices to give us tags to work with throughout the week. These will be used later for real time values, history, and alarms.

To create our first simulated device, follow these steps.

1. If necessary, log into the Gateway web page.
2. Click **Config**.

The screenshot shows the Ignition Configuration interface. The left sidebar is titled 'Config' and contains several sections: SYSTEM (Overview, Backup/Restore, Ignition Exchange, Licensing, Modules, Projects, Redundancy, Gateway Settings), NETWORKING (Web Server, Email Settings, Gateway Network), SECURITY (General, Auditing, Users, Roles, Service Security, Identity Providers, OAuth2 Clients, Security Levels, Security Zones), DATABASES (Connections, Drivers, Store and Forward), and ALARMING (General). A search bar is at the bottom of the sidebar. The main content area is titled 'Configuration' and includes a green banner stating 'Trial Mode 0:49:34 We're glad you're test driving our software. Have fun.' Below this, there are six sections: 'PLATFORM' (Update System Name, Configure Redundancy, Install or Upgrade a Module, Create New Project, Activate a License, Download Gateway Backup), 'NETWORKING' (Change Web Server Settings, Enable SSL for the Gateway Network, Create an Email Profile, Manage incoming/outgoing Gateway Network connections), 'SECURITY' (Change General Gateway Security Settings, Create a new user, Assign a user a new role, View the logs of an audit profile, Define a Security Zone, Set access levels on a Security Policy), 'CONNECTIONS' (Create a new database connection, Connect to a 3rd party OPC server, Create a new device connection), 'SYSTEMS' (Create an alarm journal profile, Manage schedules and holidays, Create a new alarm notification profile, Test an alarm notification pipeline, Add users to an on-call roster), and 'DATA ACQUISITION' (Define a new realtime tag provider, Manage tag historians, Quickly read or write tags in a device).

3. Click **Device Connections**. You may need to scroll down to see it under the OPC UA heading.



4. Click **Create new Device...**

Name	Type	Description
No Devices		

Create new Device...

5. Select **Programmable Device Simulator**. You may need to scroll down.

6. Click **Next**.

- Omron NJ Driver
Connect to Omron NJ series PLCs.
- Programmable Device Simulator
A simulator device that can be configured with a user-defined hierarchy of static or function-driven values.
- Siemens S7-1200
Connect to Siemens S7-1200 PLCs over Ethernet.
- Siemens S7-1500
Connect to Siemens S7-1500 PLCs over Ethernet.
- Siemens S7-300
Connect to Siemens S7-300 PLCs over Ethernet.
- Siemens S7-400
Connect to Siemens S7-400 PLCs over Ethernet.
- TCP Driver
- UDP Driver

Next >

7. Enter **DairySim** in the Name field.

8. Click **Create New Device**.

General	
Name	DairySim
Description	DairySim
Enabled	<input checked="" type="checkbox"/> (default: true)

Simulator Program Settings	
Repeat Program	<input checked="" type="checkbox"/> (default: true)
Base Rate (ms)	1000 (default: 1,000)

Create New Device

The device is created.

Create a Second Simulated Device

Now, we are going to repeat the steps above to create a second device simulator.

You should already be on the devices page.

1. Click **Create new Device...**

✓ Successfully created new Device "DiarySim"

Name	Type	Description	Enabled	Status	More ▾	edit
DiarySim	Programmable Device Simulator		true	Running		

→ Create new Device...

2. Select **Programmable Device Simulator**. You may need to scroll down.

3. Click **Next**.

4. Enter **GenSim** in the Name field.

5. Click **Create New Device**.

General	
Name	<input type="text" value="GenSim"/>
Description	<input type="text"/>
Enabled	<input checked="" type="checkbox"/> (default: true)

Simulator Program Settings	
Repeat Program	<input checked="" type="checkbox"/> (default: true)
Base Rate (ms)	<input type="text" value="1000"/> (default: 1,000)

Create New Device

The second device simulator is created.

Loading the Programs

Now we can set the program for the simulators.

1. Click **More** to the right of DairySim.
2. Click **edit program**.

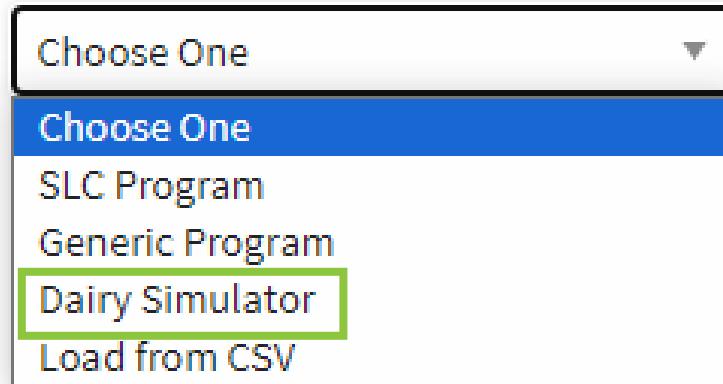
✓ Successfully created new Device "GenSim"

Name	Type	Description	Enabled	Status	
DiarySim	Programmable Device Simulator		true	Running	<button>More ▾</button> <button>edit</button>
GenSim	Programmable Device Simulator		true	Running	<button>More ▾</button> <button>edit</button>
→ Create new Device...					delete

3. Select **Dairy Simulator** from the dropdown list.

DairySim

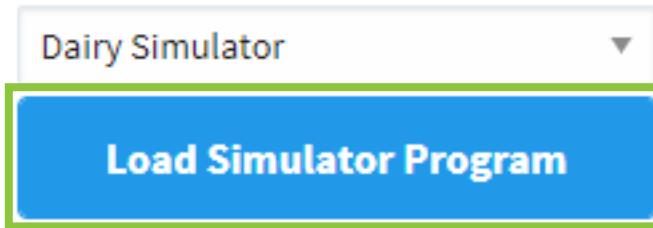
Load Program



4. Click **Load Simulator Program**.

DairySim

Load Program



5. Scroll to the bottom of the page.

6. Click **Save Program**.

A screenshot of a table-based configuration interface for a program. It has two rows of data:

0	Overview/Motor 8/Amps	realistic(100.0, 1.2, 0.06, 0.25, true)	Double	Remove
0	Overview/Motor 8/HOA	0	Int16	Remove

Below the table is a row of buttons: "Id Instruction", "Save Program" (highlighted with a green border), and navigation arrows "<<< 1 2 3 4 >>>".

You will be returned to the Devices page. A **Successfully loaded program** message will appear.

Now, we will repeat those steps for the other simulator.

1. Click **More** to the right of DairySim.
2. Click **edit program**.

Successfully reloaded program				
Name	Type	Description	Enabled	Status
DiarySim	Programmable Device Simulator		true	Running
GenSim	Programmable Device Simulator		true	Running

→ Create new Device...

3. Select **Generic Program** from the dropdown list.

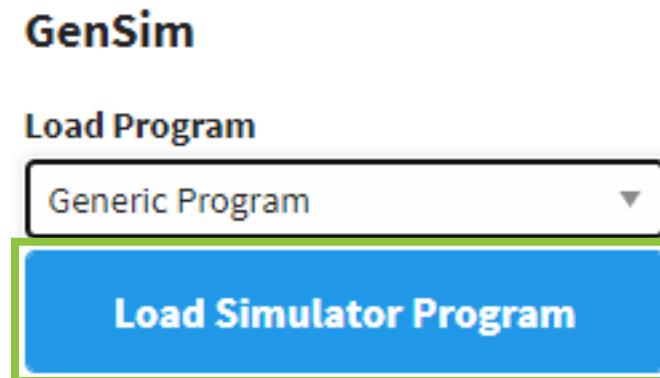
GenSim

Load Program

Choose One ▾

- Choose One
- SLC Program
- Generic Program**
- Dairy Simulator
- Load from CSV

NO PROGRAM LOADED

4. Click **Load Simulator Program.**

5. Scroll to the bottom of the page.

6. Click **Save Program.**

0	ReadOnly/ReadOnlyBoolean1	readonly(false)	Boolean	Remove
0	ReadOnly/ReadOnlyBoolean2	readonly(false)	Boolean	Remove
0	ReadOnly/ReadOnlyDouble1	readonly(1.1)	Double	Remove
0	ReadOnly/ReadOnlyDouble2	readonly(1.2)	Double	Remove
0	ReadOnly/ReadOnlyFloat1	readonly(1.1)	Float	Remove

Add Instruction << < 1 2 3 > >>

Save Program

Both devices are now loaded and ready to use.

Connecting to a Database

Many features of Ignition require a connection to an external database. Later in this course, we will explore several of these features, but we will go ahead and set up the database connection now.

Ignition is capable of connecting to any SQL-based database, so long as a JDBC driver is available.

While we are only going to create a single database connection, Ignition supports unlimited connections. You can even have connections to different types of databases within the same Gateway.

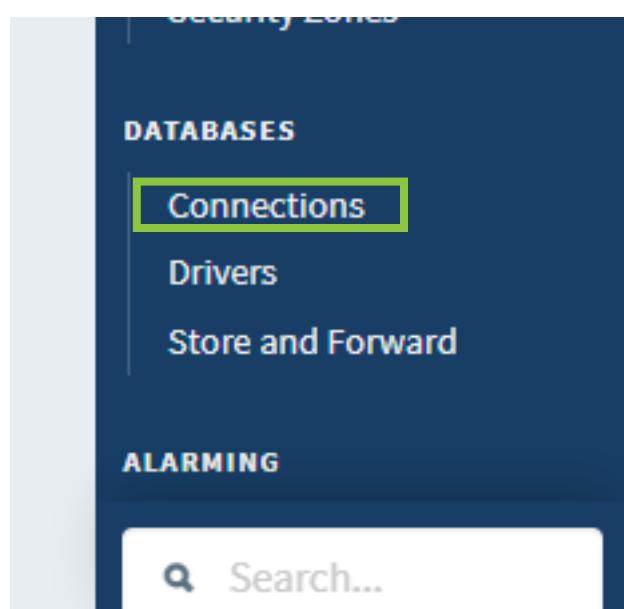
Connecting to Microsoft SQL Server

For this course, we will use Microsoft SQL Server as our database, which is already installed on the computers.

Why SQL Server?

We use Microsoft SQL Server in this class because all of the drivers needed to connect to it are included in a default Ignition installation. This is not the case with the other highly popular database server, MySQL. We are not expressing a preference for one over the other, but rather, going with the one with simpler setup.

1. Click **Connections** under the Databases section. You may need to scroll.



2. Click **Create new Database Connection...**

Name	Description	JDBC Driver	Translator	Status
No Database Connections				
Create new Database Connection... <p>Note: For details about a connection's status, see the Database Connection Status page.</p>				

3. Click **Microsoft SQL Server**.4. Click **Next**.

- MariaDB**
The MariaDB (a community-owned fork of MySQL) JDBC Driver - compatible with all MariaDB servers, MySQL 5.x (>= 5.5.3) and MySQL 8.0.
- Microsoft SQLServer**
The Microsoft SQL Server JDBC Driver is a Java Database Connectivity (JDBC) 4.2 compliant driver.
- MySQL**
The official MySQL JDBC Driver, Connector/J.
- Oracle Database**
The Oracle Database JDBC driver.
- PostgreSQL**
The official PostgreSQL JDBC Driver.
- SQLite**
Driver for the popular embedded database system.

Next >

5. Enter **DB** in the Name field.6. Edit the Connect URL field to read **jdbc:sqlserver://localhost\SQLEXPRESS**.

Note: These settings may vary in an in-person class versus a virtual class. Check with your instructor.

7. Enter **sa** in the Username field.

8. Enter the password provided by your instructor in both password fields.

Main Properties	
Name	<input type="text" value="DB"/> Choose a name for this database connection.
Description	<input type="text"/>
JDBC Driver	Microsoft SQLServer ▾ The JDBC driver dictates the type of database that this connection can connect to. It cannot be changed once created.
Connect URL	<input type="text" value="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: <code>jdbc:sqlserver://host\instanceName[:port]</code> 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 <i>database name</i> to connect to using the <code>databaseName</code> property in the <i>Extra Connection Properties</i> .
Username	<input type="text" value="sa"/>
Password	<input type="password"/> Re-type password for verification.
Password	<input type="password"/>

9. Scroll down.

10. Click **Create New Database Connection**.

Failover Mode	<input type="text" value="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	<input type="text" value="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)
Validation Timeout	<input type="text" value="10000"/> The time in milliseconds between database validation checks. (default: 10,000)

Show advanced properties

Create New Database Connection

Launch Designer and Create a Project

Now that we have set up some connections, we'll launch the Designer and get started creating a project. Ignition has a special launcher program available in the Gateway that allows us to use the Designer without first installing Java on any computer.

Once the Designer is open, we will familiarize ourselves with the panels and workspaces that are available. Configuration for almost every module is available through the Designer, and this is where we will be spending most of our time throughout the week working on our projects.

Installing the Designer Launcher

The Designer Launcher is a free tool, which can be downloaded directly from the Gateway web page.

1. Open the **Gateway web page**. Remember, you can type <http://localhost:8088> in your browser.
2. Click **Download Designer Launcher**.

Get the Designer

The Designer brings all your data, systems, and developers together into one beautifully simple, integrated development environment specifically designed to help you build industrial applications more quickly.

[Download Designer Launcher](#)

3. Click **Download for Windows**



Download the Designer Launcher

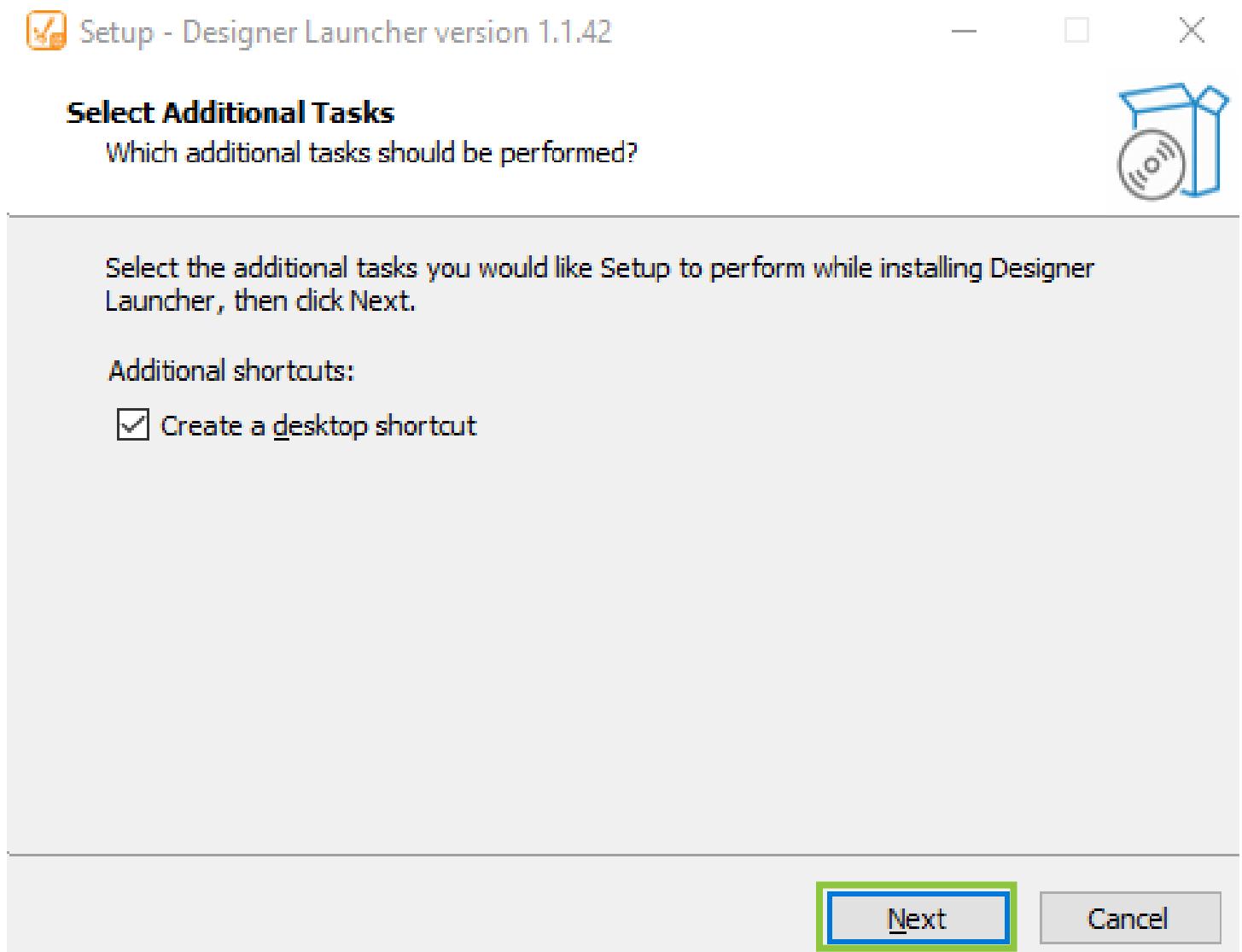
[Download for Windows](#)

We've detected you're on Windows. Download the Designer Launcher for Windows and follow these steps below to install.

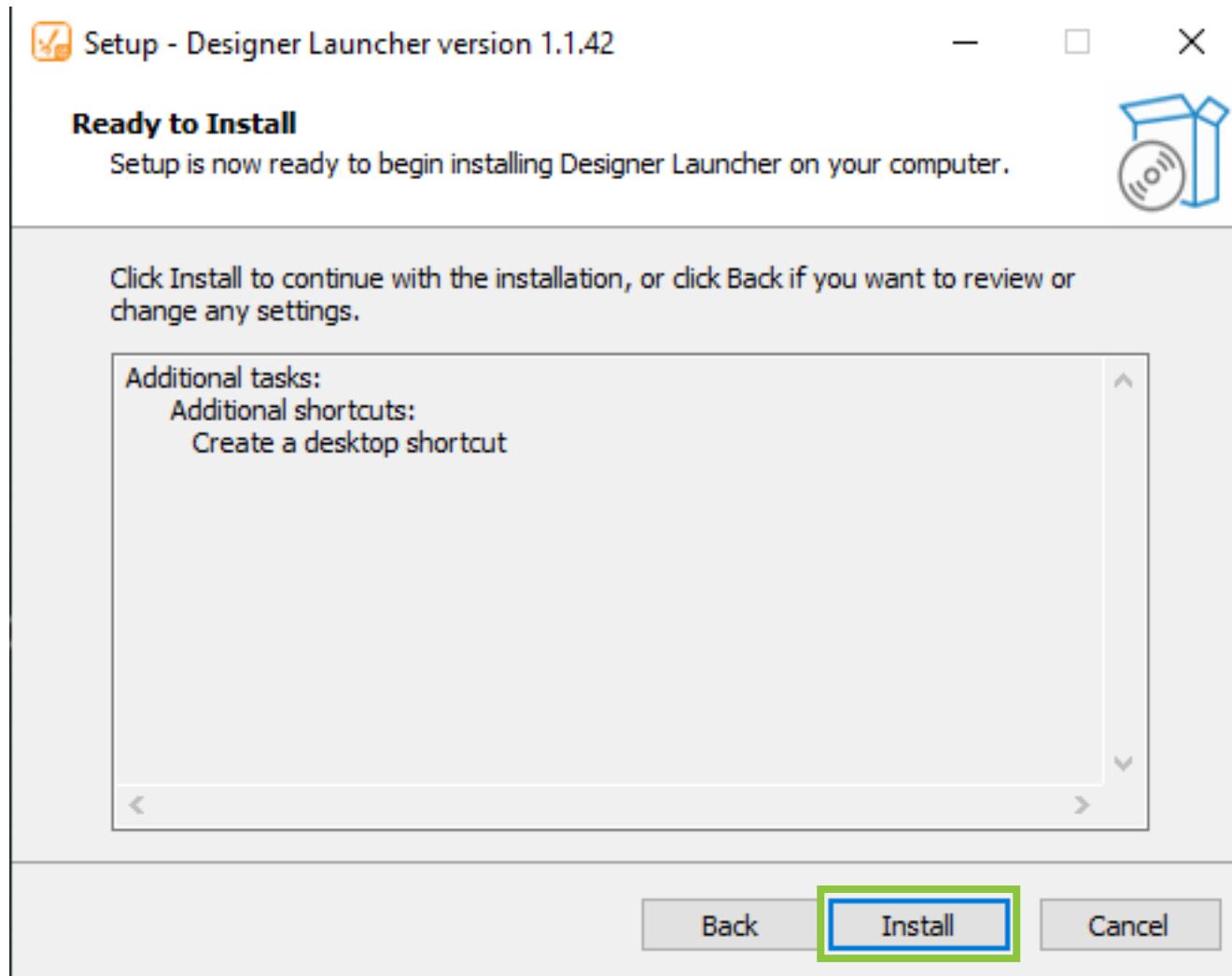
The installer program will download. Depending on your internet connection, it might take a few moments.

4. Double-click the **installer**. It should be in your **Downloads folder**.

5. Click **Next**. You can choose whether or not you want a desktop shortcut.

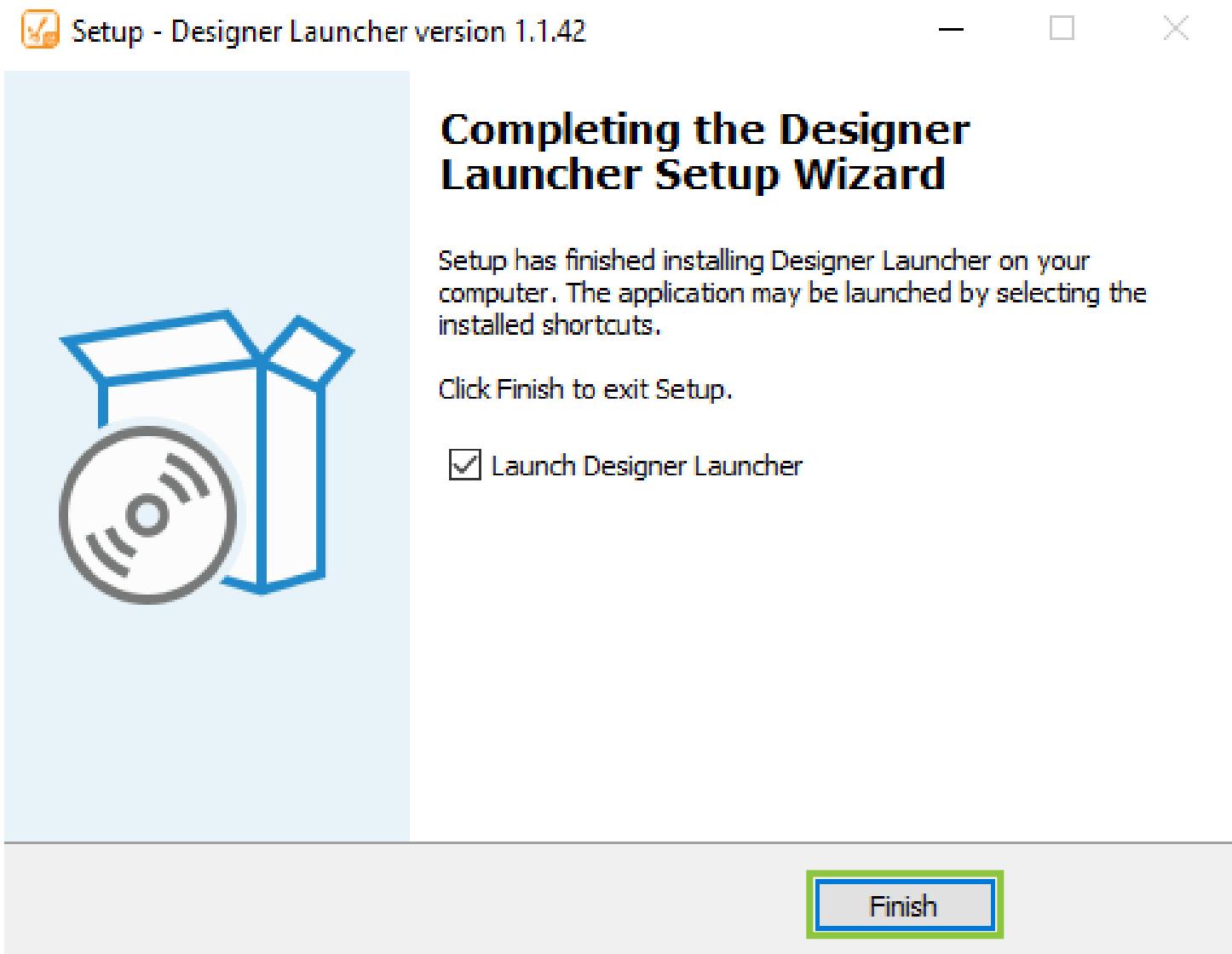


6. Click **Install**.



The launcher installs.

7. Click **Finish**. Make sure **Launch Designer Launcher** is checked.

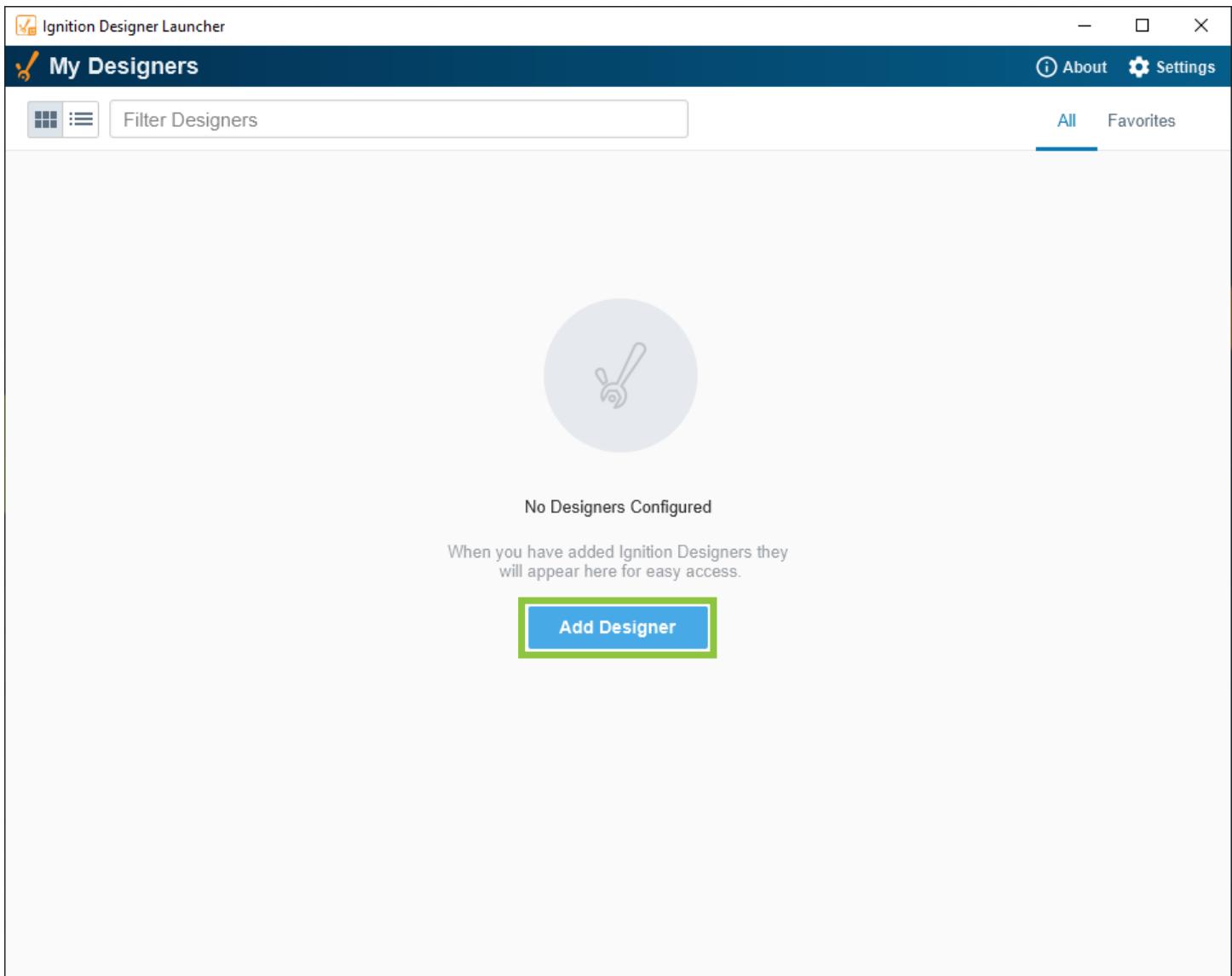


Adding and Opening a Designer

The Launcher needs to be connected to a Gateway in order to add Designers. Remember, a Designer is launched from a Gateway.

Your Gateway should automatically appear in the Designer Launcher interface. In a production environment, it is possible that you will have connections to multiple Gateways, so you would need to select the Gateway that you want this Designer to be launched from.

1. Click **Add Designer**.



2. Click **your Gateway** from the list.

Your Gateway may be listed twice—once by its IP address and once by its hostname (localhost). It does not matter which one you select.

Note: In the classroom, you may see all of your classmates' Gateways, as well as your instructor's, listed as well.

3. Click **Add Designer**.

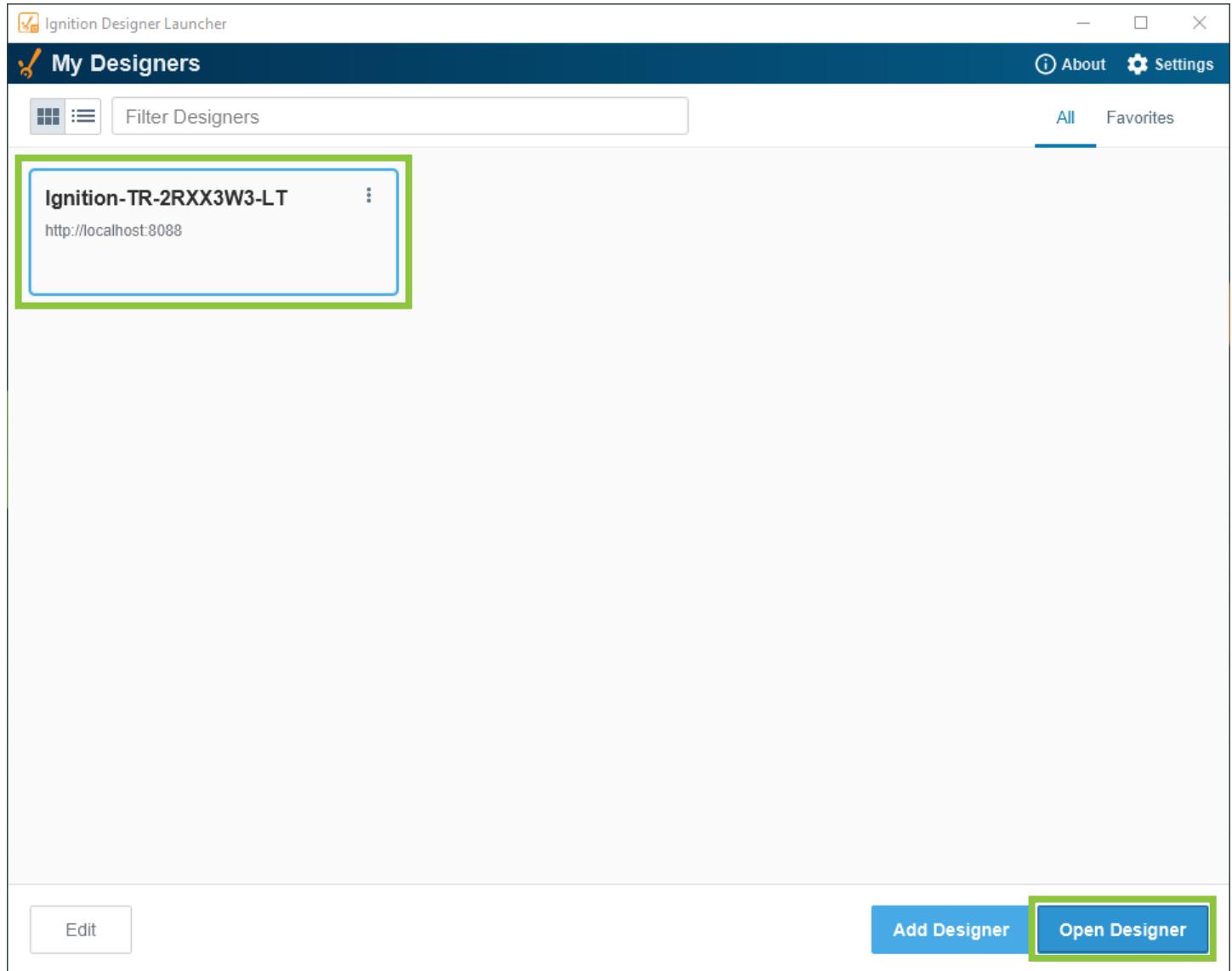
The screenshot shows a software window titled "Add Designer". At the top, there are tabs for "On Your Network" (which is selected) and "Manual". Below the tabs is a search bar labeled "Filter Gateways". A table lists two gateways:

Name	URL	Version	OS	System	Status
Ignition-TR-2RXX3W3-LT	http://172.24.160.1:8088	8.1.42		Independent	RUNNING
Ignition-TR-2RXX3W3-LT	http://localhost:8088	8.1.42		Independent	RUNNING

Below the table, a message says "Scanning local network for available gateways...". At the bottom right are "Cancel" and "Add Designer" buttons, with "Add Designer" being highlighted by a green border.

4. Click **the card for your Gateway**.

5. Click **Open Designer**.



The Designer will take a moment to launch. It will ask you to log in.

6. Enter **admin** as the username.
7. Enter **password** as the password.

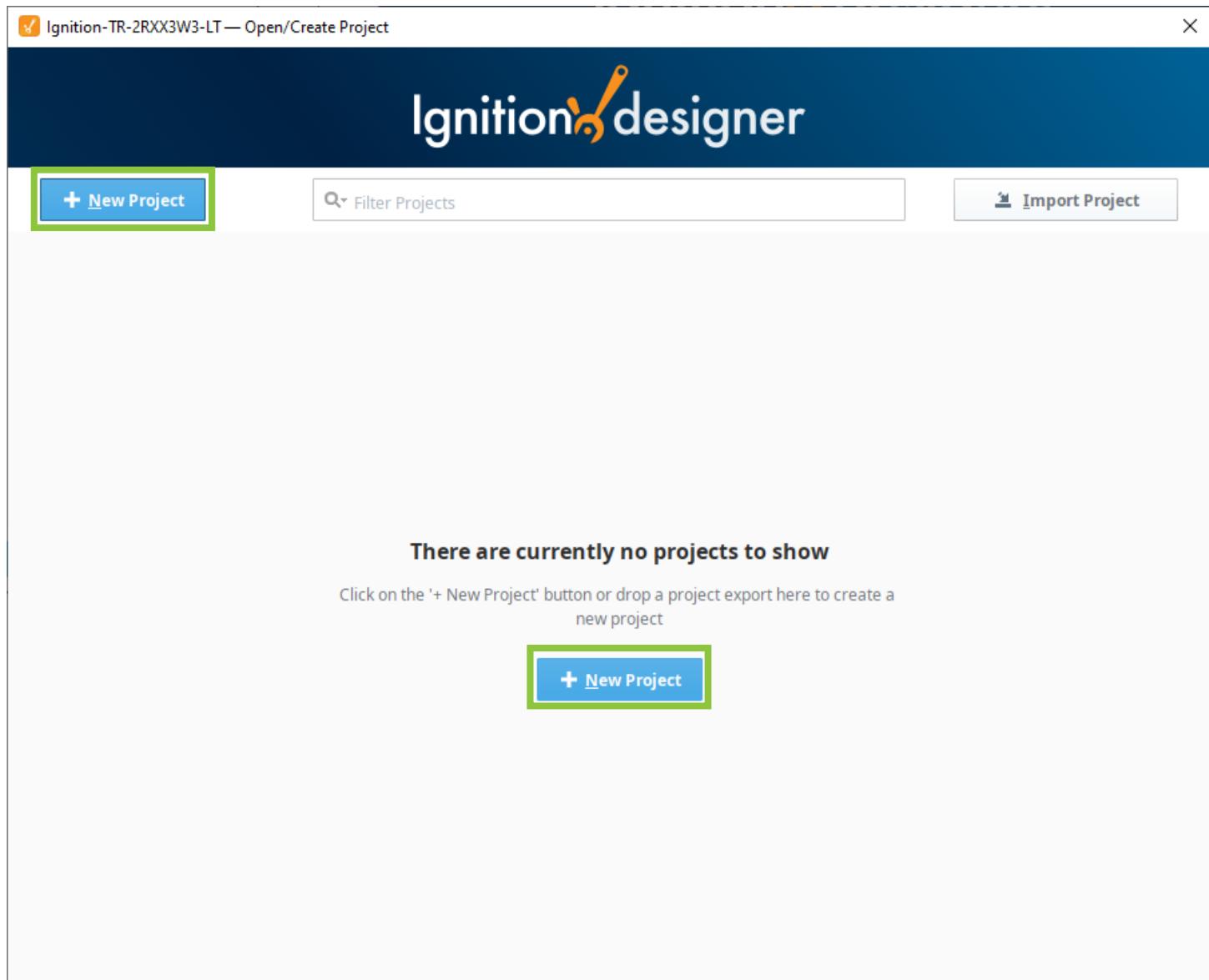
8. Click **Login**.

Creating a New Project

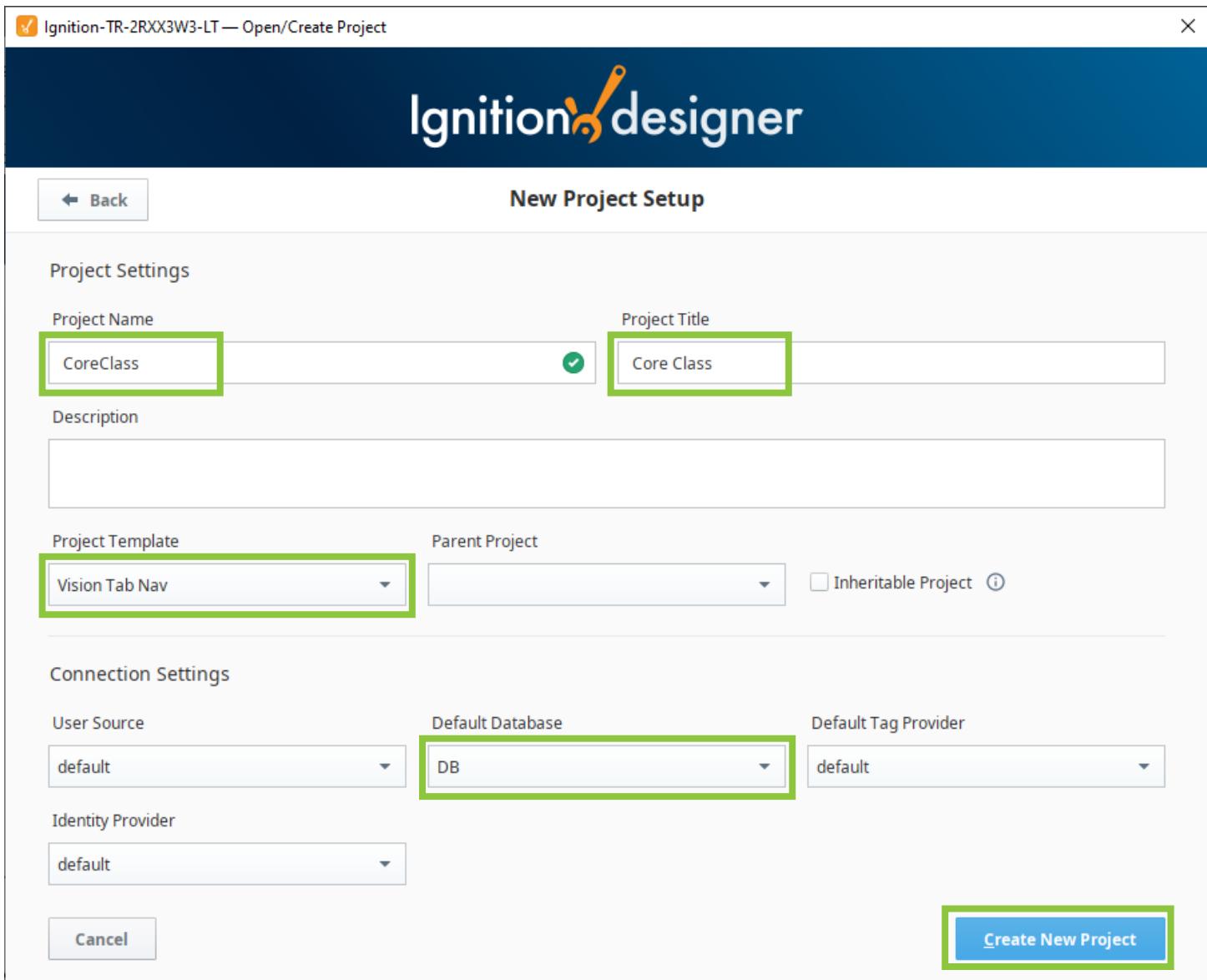
Projects are where we do most of our work in Ignition. Projects will contain all of the resources we need, and are the space in which we design our interfaces, work with tags and alarms, and much more.

Projects can be created in either the Gateway or Designer. For our purposes, we are going to use Designer.

1. Click either of the **+ New Project** buttons.



2. Enter **CoreClass** in the **Project Name** field. Be sure to not put a space in the name.
3. Enter **Core Class** in the **Project Title** field. A space is fine here.
4. Select **Vision Tab Nav** from the **Project Template** list.
5. Ensure that **DB** is selected in the **Default Database** list.
6. Click **Create New Project**.



The Project Name is used internally by Ignition to identify this project. As such, non-alphanumeric characters such as spaces are not allowed. The Project Title, on the other hand, is intended as a human-friendly identifier for the project, so any characters are allowed.

Project templates provide a convenient starting point for projects. The Vision Tab Nav template we will use includes a set of pre-built Vision Windows, allowing us to get started designing faster.

Each project can have a set of default connections to user sources, databases, tag providers, and identity providers. None of these limit what the project can do; they are only defaults. For example, a project can be set up to connect to any number of databases, but the default database will be pre-selected in dialog boxes and other interface elements within Designer.

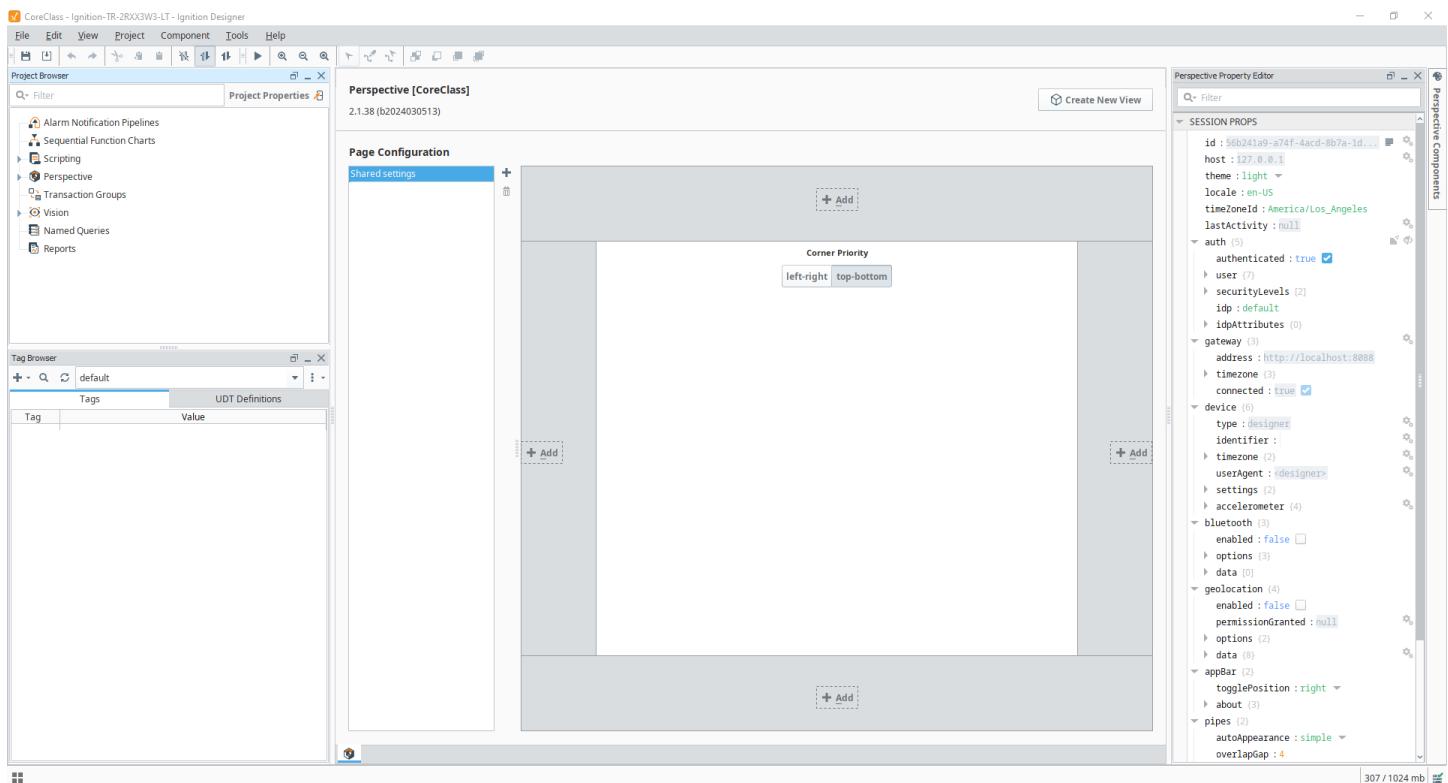
Introduction to Designer

Now that we have Designer installed and a project created in it, we want to take a look at it. We will explore its interface, look at the tools it provides, and discuss a bit about how to customize its look and feel.

The Designer Interface

The Ignition Designer is where the majority of configuration and design work is done. The Designer uses web-launch technology to open and edit your projects. This is how you can configure your Perspective and Vision projects.

The Designer provides all the firepower to bring your projects to life. You can create user interfaces by dragging Perspective components onto a view, Vision components onto a window, and Tags onto your components to instantly bind data to tables, charts, and graphs. You can set up Tag History and Transaction Groups to log data to your databases. You can create Reports to print data. The Designer saves all your projects to the Gateway so everything is controlled in one place.



Panels

The main interface in Designer is made up of a series of panels. Each panel serves a specific purpose. Panels can be moved around on the screen, and individually opened and closed as desired. If you are working with multiple monitors, you can even move panels to other monitors to maximize your screen space.

There is also a toolbar across the top of the screen. We will explore toolbar buttons as we need them.

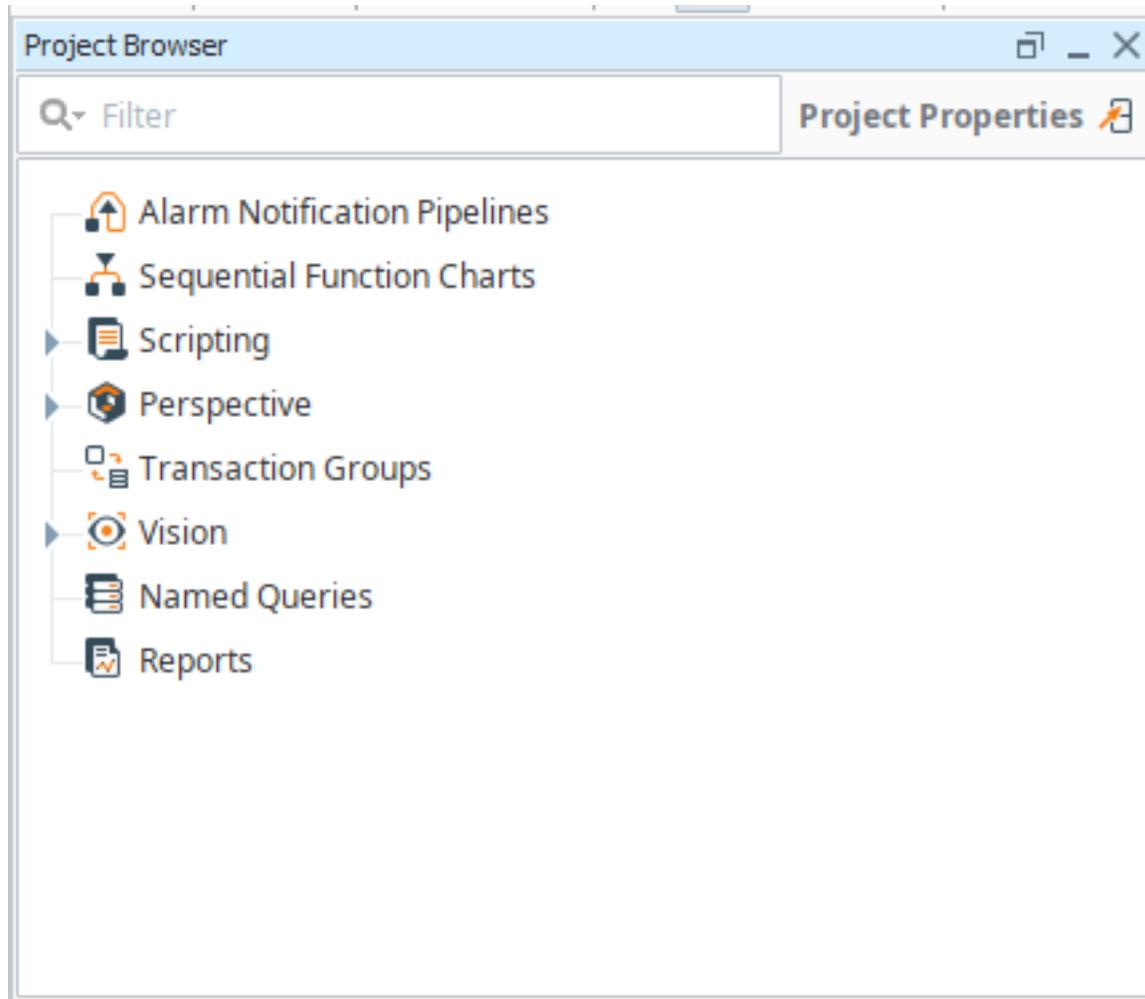
Project Browser

The set of panels that are visible by default depends on which module you currently have active, but all modules always display the Project Browser. By default, it is always in the top left corner of the screen.

The Project Browser lists the modules you currently have installed. The license we are using for class includes the Alarm Notification Pipelines, Sequential Function Charts, Scripting, Perspective, Transaction Groups, Vision, Named Queries, and Reports modules. In this class, we will end up exploring all of those except Sequential Function Charts and Named Queries.

In your work environment, you may have more or fewer modules.

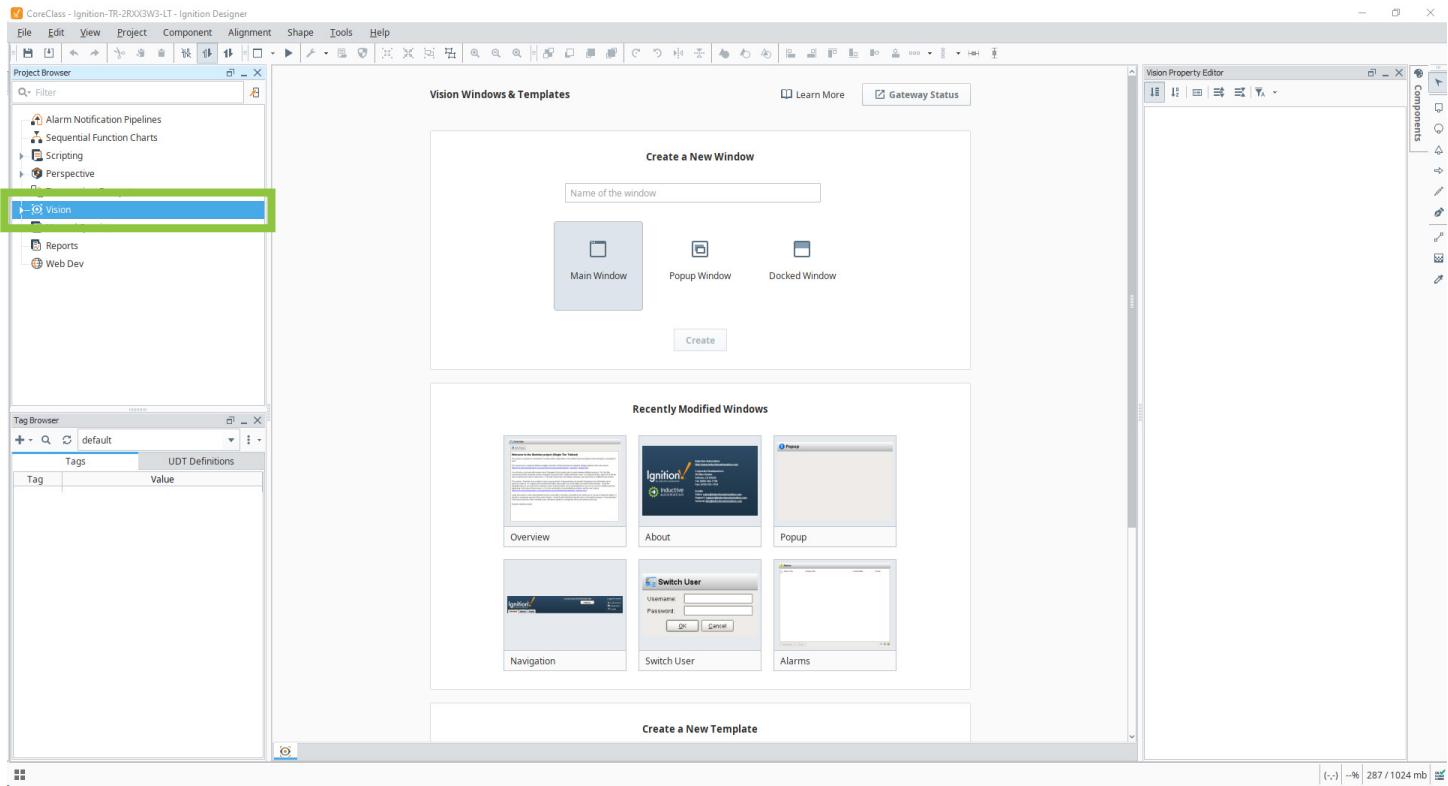
The Project Browser can also show details within each module, allowing you to select and at times configure the individual elements within a module. Any module that has a small triangle to the left of its name has sub-elements.



Vision Default Panels

When we switch to the Vision module, we get a set of default panels for building Vision clients. We will use this module to explore customizing the workspace.

1. Click **Vision** in the Project Browser panel.



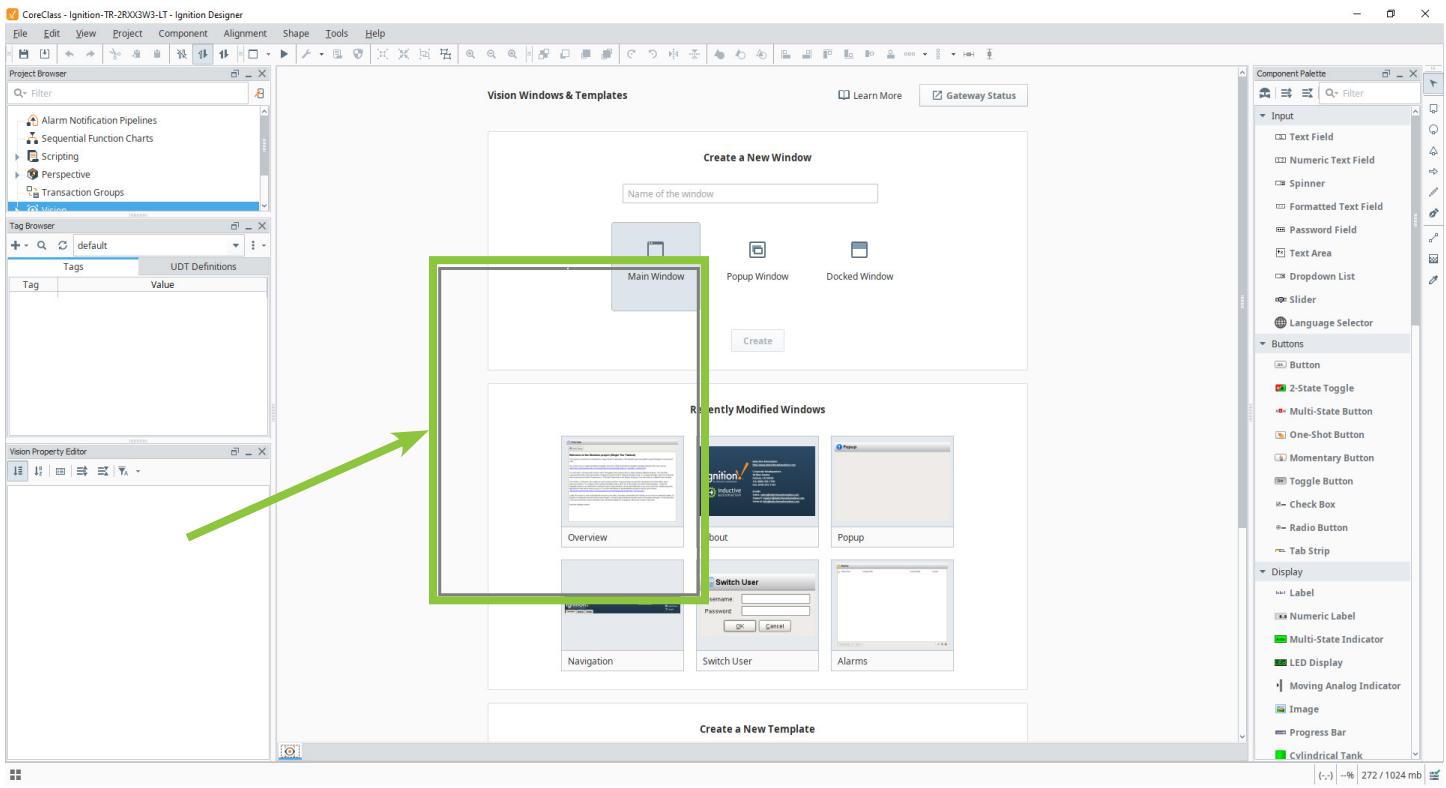
Designer updates to display the Vision module's default panels.

You will see that by default, Vision has the **Project Browser** in the top left corner. Below that is the **Tag Browser**, which will be used to manage and view tags, and then below that is the **Vision Property Editor**. On the right side of the screen is the **Component Palette**, and to its right is the **Vision Drawing** toolbar.

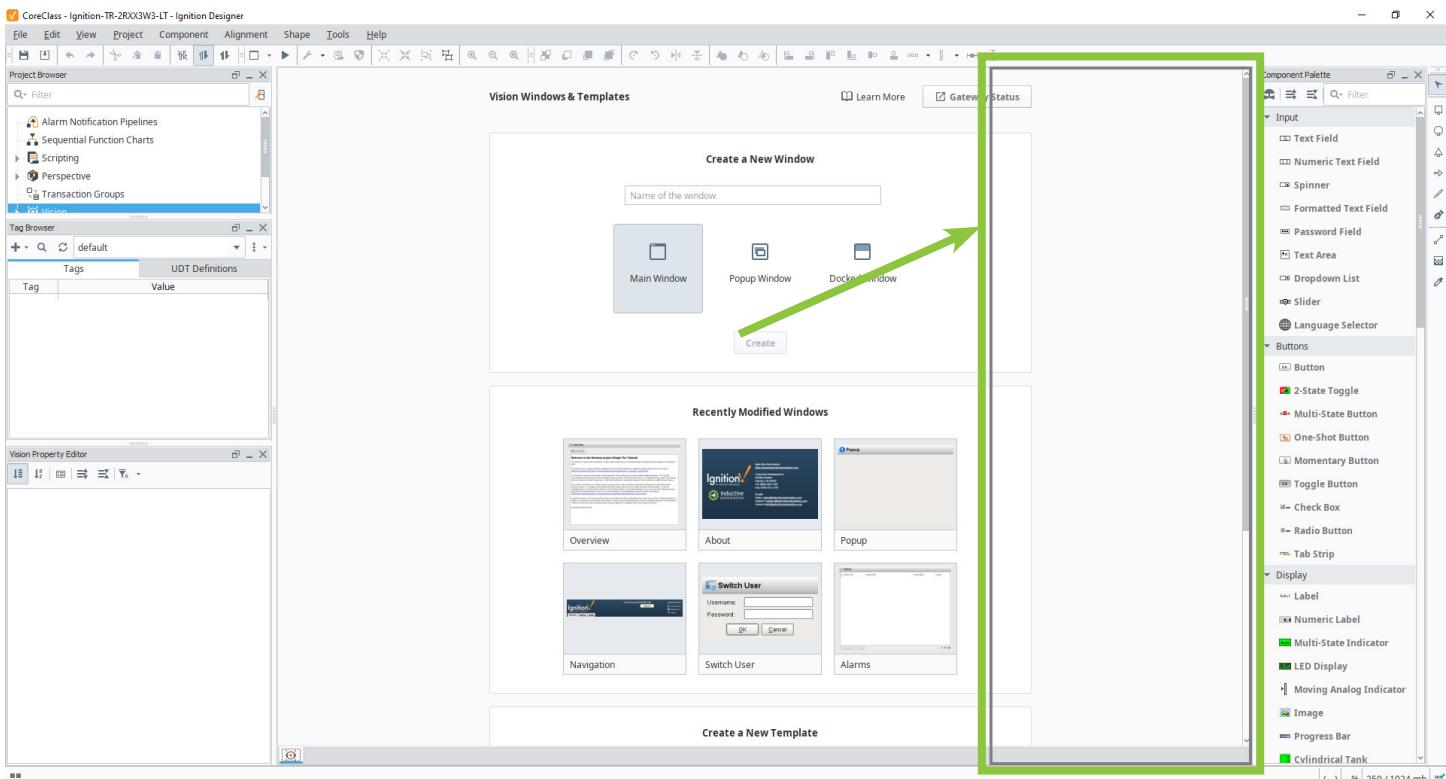
Customizing Panels

All windows, components, and templates have properties that can be edited. Many components have dozens of properties. Thus, it can become inconvenient to have the Vision Property Editor taking up a small space in the bottom left corner of the screen. To demonstrate how to customize the interface, we are going to move the Vision Property Editor to the right side of the screen. We will also collapse the Components Palette so that it takes up less space.

1. **Press and hold your mouse button** on the top of the Vision Property Editor, directly on its title bar.
2. **Drag** the editor towards the middle of the screen. It will appear as a gray outline.

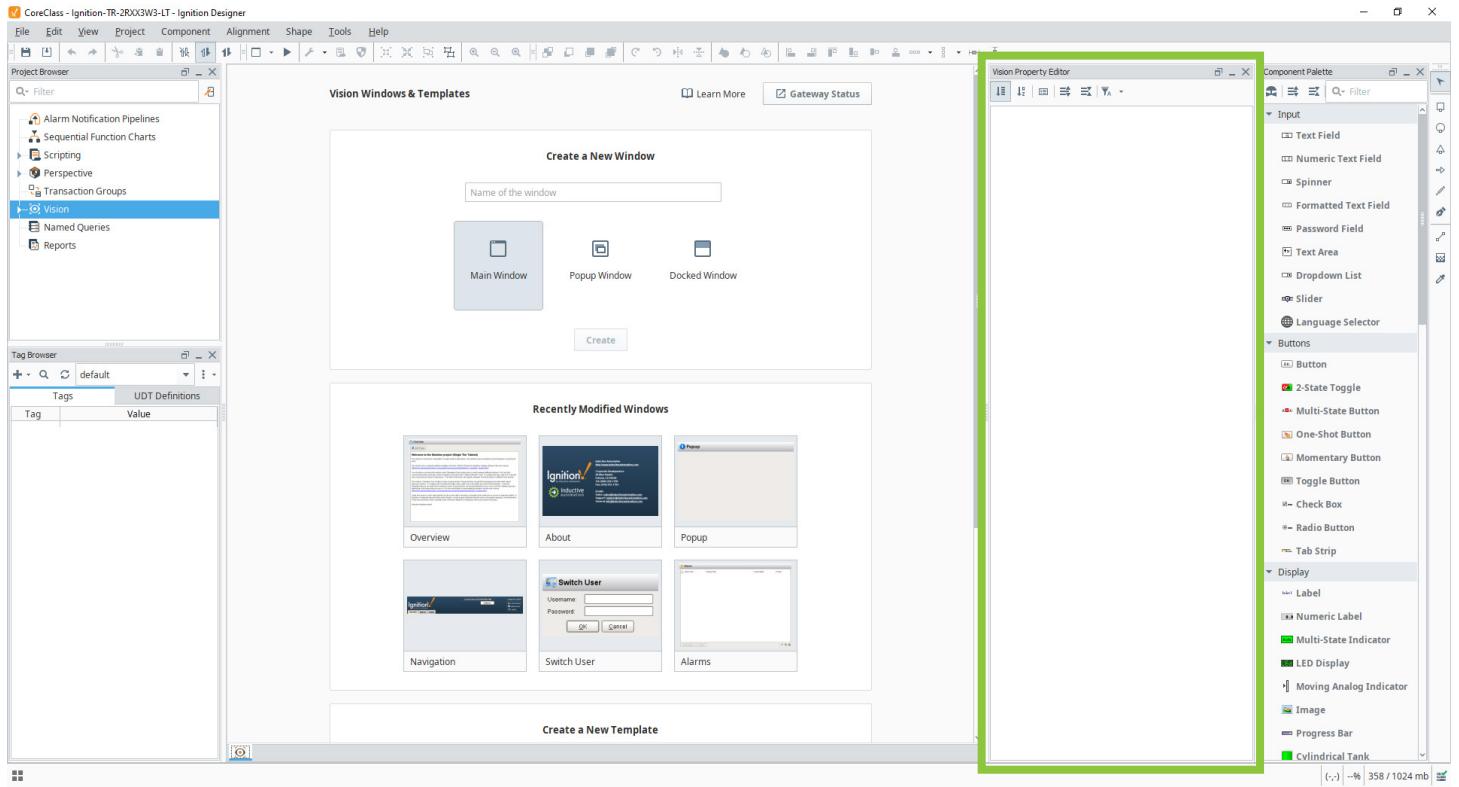


3. Continue dragging until a dark gray outline appears to the **left of the Component Palette**.



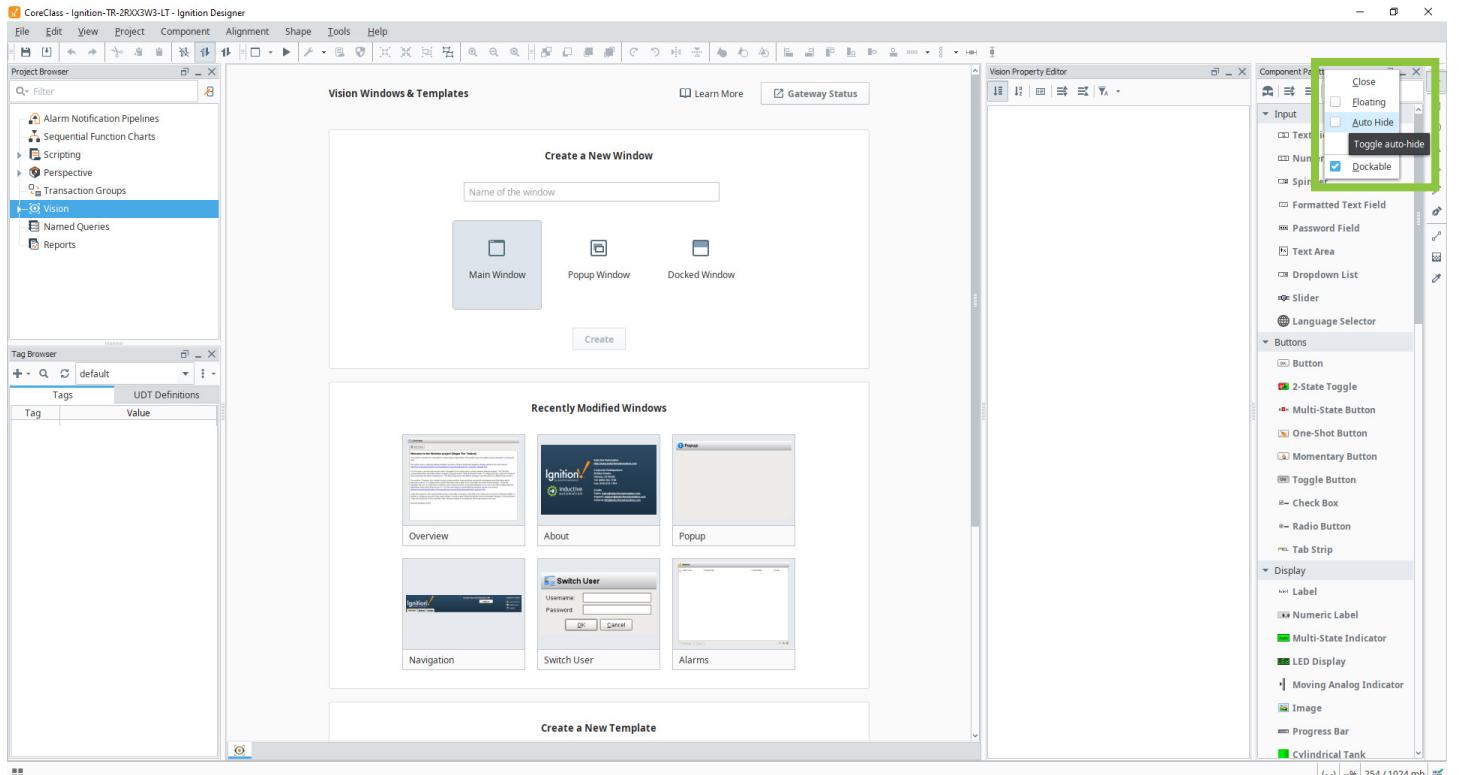
Ch 5. Introduction to Designer

4. Release your mouse to dock the panel on the right side of the screen.



5. Right-click on the words **Component Palette**.

6. Click **Auto-hide**.



At any point, you can access the panel by moving your mouse over the word **Components**.

Moving Panels

You can also move panels without docking them. This is helpful in situations where you wish to move panels to another monitor.

To move a panel without docking it, simply drag it by its header, just as we did with the Vision Property Editor. As long as you do not release your mouse near one of the edges of the Designer window, the panel will not be docked. You can also press and hold the **Ctrl** key on your keyboard to prevent the panel from docking, even if you are near one of the edges of Designer.

Closing Panels

Every panel has a close button—a small **x** in the top right corner—that you can use to close the panel if you decide you do not need it.

Reopening Panels

If you close a panel and then decide you need it after all, you can reopen a panel by using the menus at the top of the screen.

1. Click **View**,
2. Click **Panels**.
3. Select the panel you wish to open. Panels will open in the same location they were in when they were closed.

Resetting Your View

You can reset the panels in any module at any time. This can be helpful if you have customized the view and then wish to return to the default, or if you have opened panels and cannot locate them because you moved them off-screen.

1. Click **View**.
2. Click **Reset Panels**.

Vision Clients, Windows, and Navigation

In chapter, we will begin designing a Vision client. We will explore how Vision clients are made up of components and templates on windows. We will also explore how to create a navigation system for the client. Finally, we will look at how to launch the client from the Designer, and how to install the Vision Client Launcher to open Vision projects directly.

Vision

Vision is one of the two primary visualization modules in Ignition. Vision projects run in the Vision Client, which is a Java-based application that looks and feels like any other installed application on a computer or other device.

Vision clients can be run on any machine that supports Java.

Like all projects, Vision clients are launched from the Gateway.

Vision Windows, Components, and Templates

Windows, components, and templates are the fundamental building blocks for projects using the Ignition Vision module. A Vision project is a collection of Windows. These windows are loaded into the Vision Client, where any number of them may be open at one time.

A window itself is a hierarchy of components. Components range in complexity from the humble Button and Label all the way to the powerful Easy Chart and Table components. Shapes such as a line or a rectangle are also considered components, but have some additional capabilities such as the ability to be rotated.

Templates are components that are reusable across many windows. They are designed separately, outside of any window. After being designed, you can add them to any window within a project. The true power of a template is that if you change its design, all uses of that template across all windows reflect those changes. This feature gives templates a major advantage over a copy-and-paste style of design.

Windows, components, and templates are designed visually with a drag-and-drop interface within the Ignition Designer. Components each have a host of properties that govern how the component looks and behaves.

Window Names and Organization

To keep your project organized, you have many options. You can create folders for each window, you can rename the folders, and you can add informational notes related to the window.

Creating Folders

By creating folders you can organize your windows. Each window must have a unique name within its folder.

Adding Notes to a Window

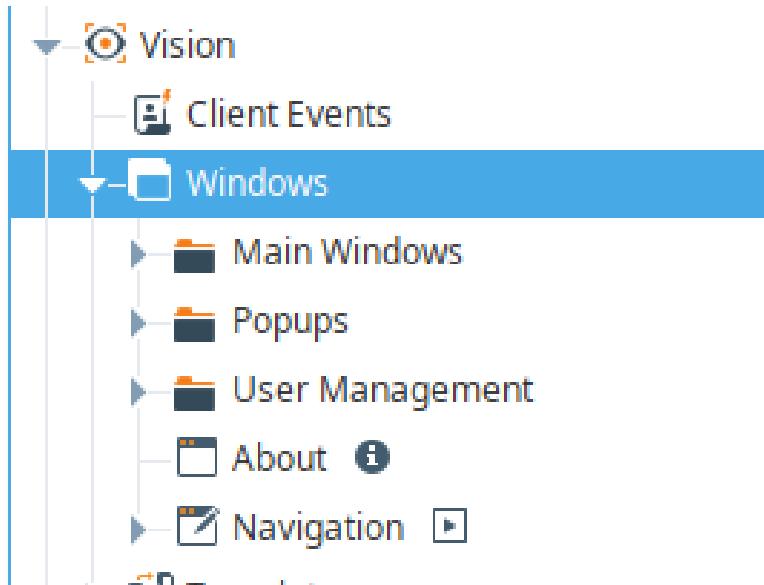
You can add notes to a window to document the purpose and any technical information about how the window works. To add window notes, right-click on a window in the Project Browser, the click Notes.

Exploring Our Project

When we created the project, we based it on the Vision Tab Nav project template. This contains a series of windows with components that provides us with a jumping off point for working in a project. By using this template, we will not have to create everything we want for our project from scratch.

Project organization

In the Project Browser, we can see that we already have three folders and two windows at the top level of our project.



You can open any of those folders to see the windows inside them.

The Main Windows folder contains three additional windows: **Alarms**, **Empty**, and **Overview**.

The Popups window contains one window, **Popup**.

The User Management window also contains a single window, **Switch User**.

Opening Windows

You can open any window by double-clicking it in the Project Browser. This will open the window in the main workspace for editing.

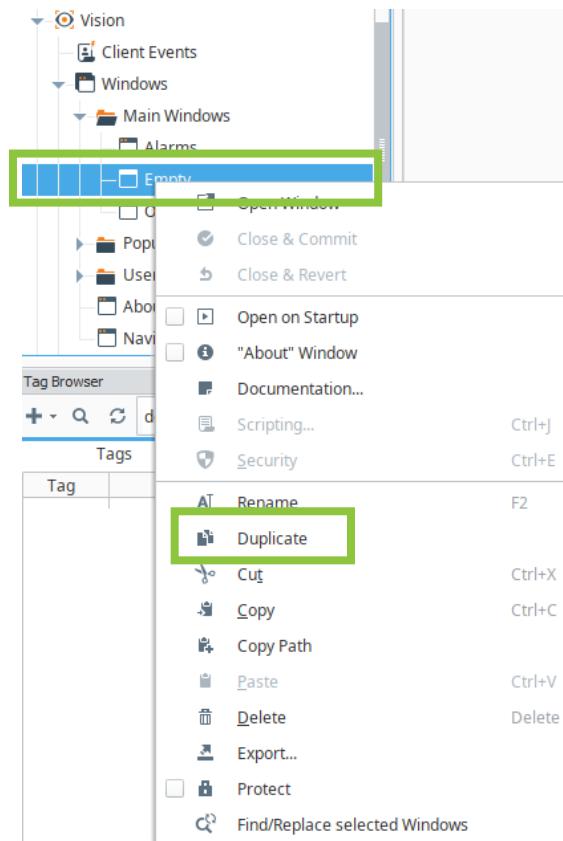
Duplicating an Existing Window

Rather than creating new windows from scratch, you can duplicate existing windows. This may save you a lot of time, as you can set up a window that has the correct properties and any shared visual elements you want to use on other windows.

Throughout the class, we will be adding additional windows to our project by duplicating the Empty window, rather than creating new windows from scratch.

To duplicate an existing window, follow these steps:

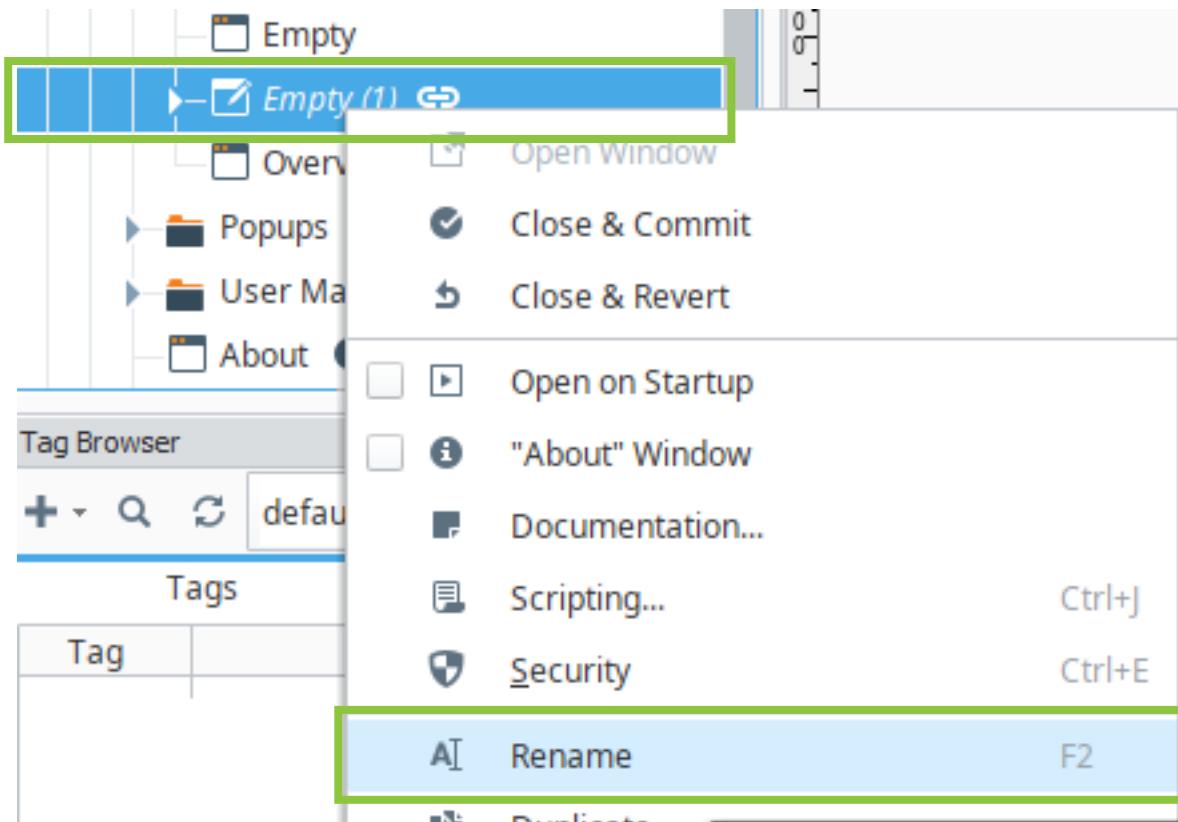
1. Right-click **Empty** in the Project Browser.
2. Click **Duplicate**.



A duplicate window, **Empty (1)** is created.

3. Right-click **Empty (1)**.

4. Click **Rename**.



5. Enter **Components**.

6. Press **Enter**.

The window is renamed.

Creating New Windows

While we will not be creating new windows from scratch, it is helpful to know how to do that.

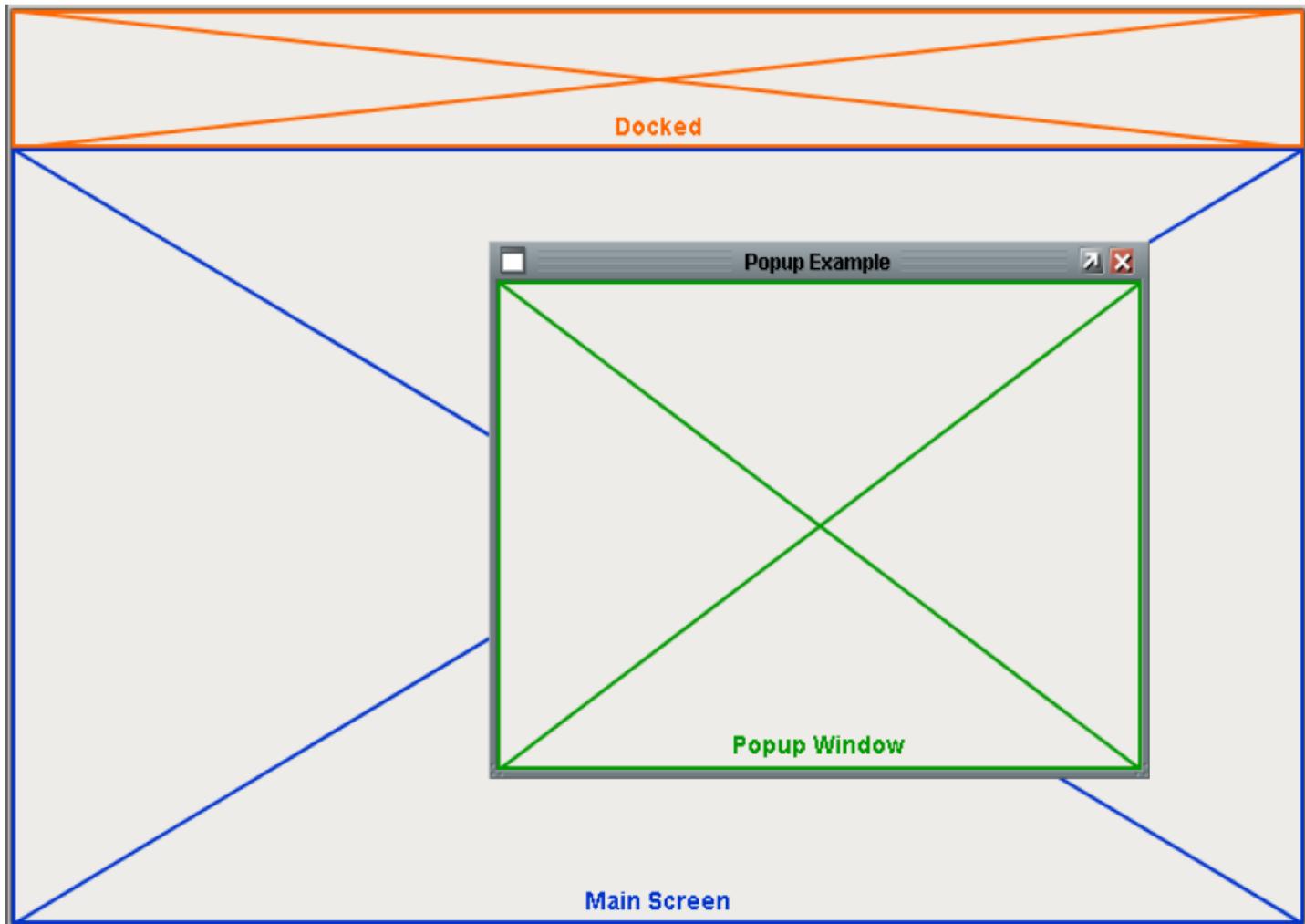
To create a new window, follow these steps.

1. Right-click **Windows** in the Project Browser.
2. Select the type of window you want to create.

Note: You can also right-click on a folder to add a new window directly to the folder.

Window Types

There are three types of windows: **Main**, **Popup**, and **Docked**.



The difference between the window types is determined by a combination of the Dock Position, Border Display Policy, Titlebar Display Policy, and Start Maximized properties. All of these can be changed on the Vision Property Editor panel.

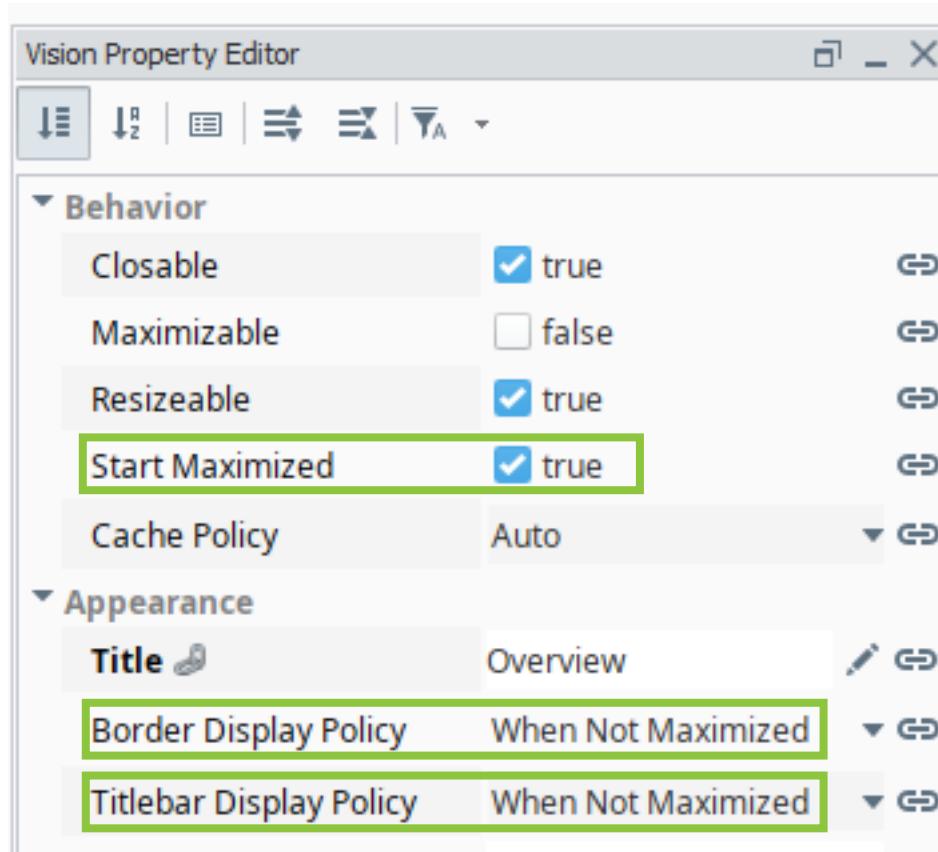
Main Windows

A main window takes up all the available space in the client and acts much like a typical “HMI screen.” There can be many main windows in a project, but you can open only one main window at any time.

To change another type of window to a main window, follow these steps:

1. **Double-click** the window in the Project Browser.
2. Set **Start Maximized** on the Vision Property Editor to **true**.

3. Set **Border Display Policy** to **When Not Maximized** or **Never**.
4. Set **Titlebar Display Policy** to **When Not Maximized** or **Never**.

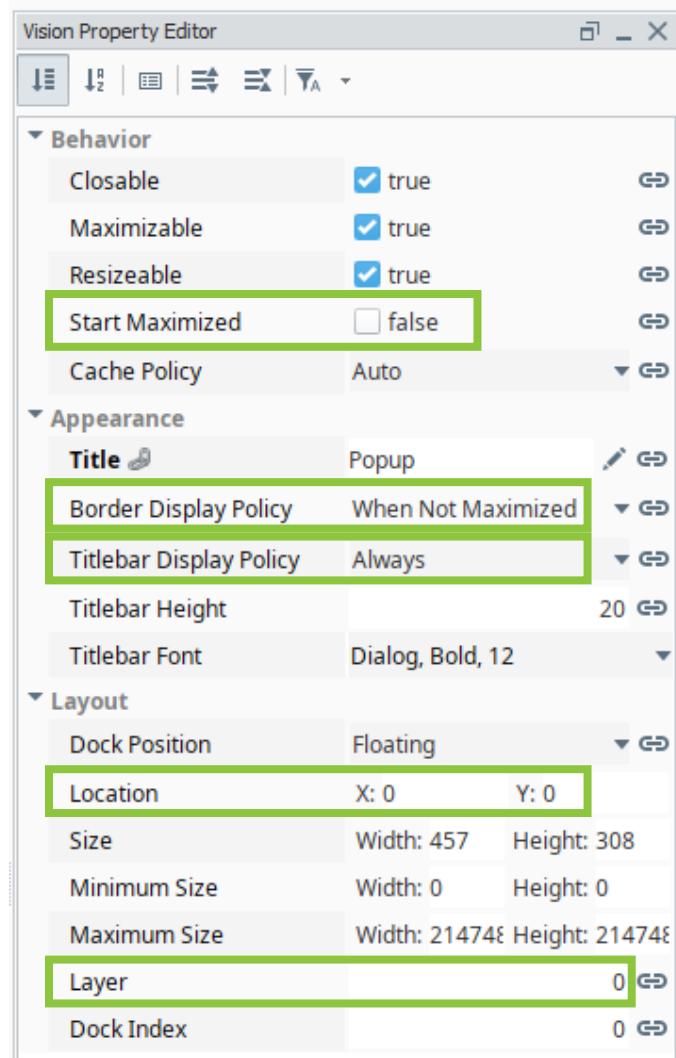


Popup Windows

The popup windows are often opened by components in a main window, and are meant to float on top of other windows. Popup windows are often parameterized so they can be reused. You define custom parameters for the popup window and then pass those new parameter values to the window when it is opened. See Chapter 18 for more details on creating and using popup windows.

To change another type of window to a popup window, follow these steps:

1. **Double-click** the window in the Project Browser.
2. Set **Start Maximized** on the Vision Property Editor to **false**.
3. Set **Border Display Policy** to **When Not Maximized** or **Always**.
4. Set **Titlebar Display Policy** to **When Not Maximized** or **Always**.
5. Set **Location** to the **x** and **y** values where you want the popup to open.
6. Set **Layer** to a non-zero number.



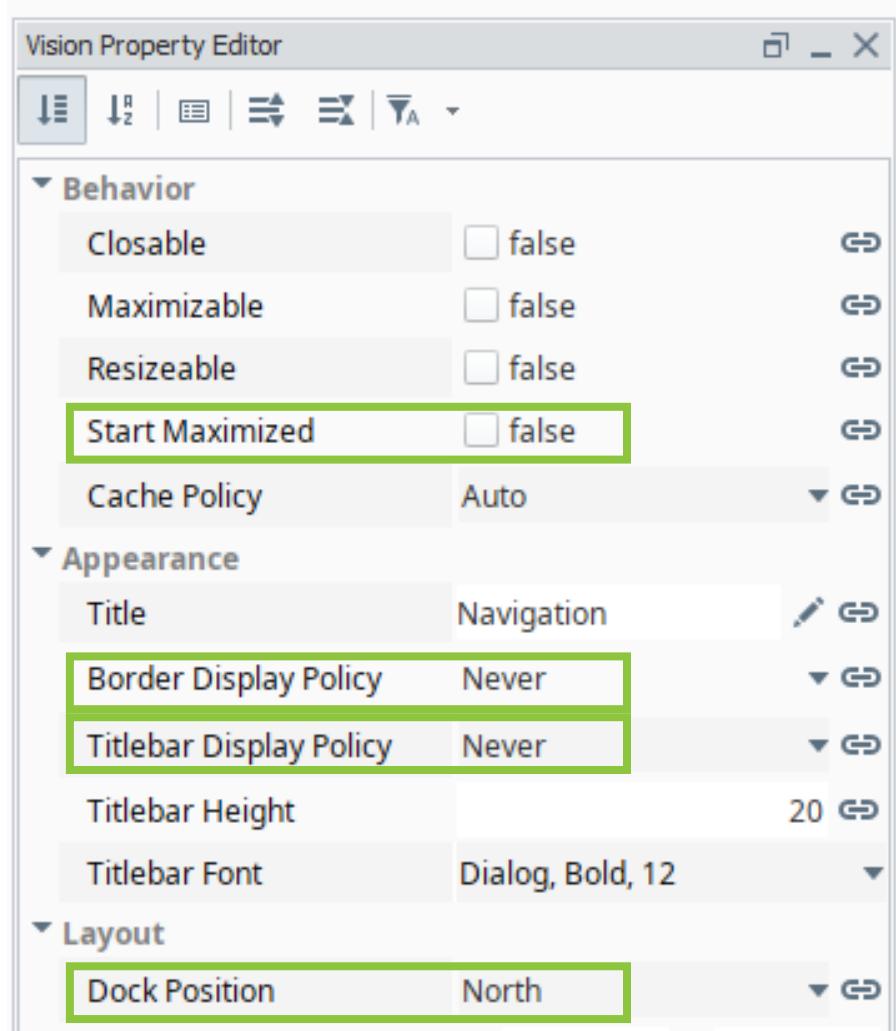
Docked Windows

A docked window will “stick” to one side of the screen, and will always be visible, even as main windows change. These screens are usually tall and skinny or short and wide, depending on the edge they’re docked to. The purpose of a docked window is to make some information always available, typically navigation controls and overall status information. Using docked windows can help eliminate repetitive design elements from being copied to each screen, making maintenance easier.

To change another type of window to a popup window, follow these steps:

1. **Double-click** the window in the Project Browser.
2. Set **Start Maximized** on the Vision Property Editor to **false**.
3. Set **Border Display Policy** to **Never**.
4. Set **Titlebar Display Policy** to **Never**.

5. Set **Dock Position** to the side of the window to which you wish to dock the window.



Renaming a Window

To rename a window, right-click on a window from the Project Browser, then click Rename. You can also use the keyboard shortcut **F2**.

Adding Windows to Navigation

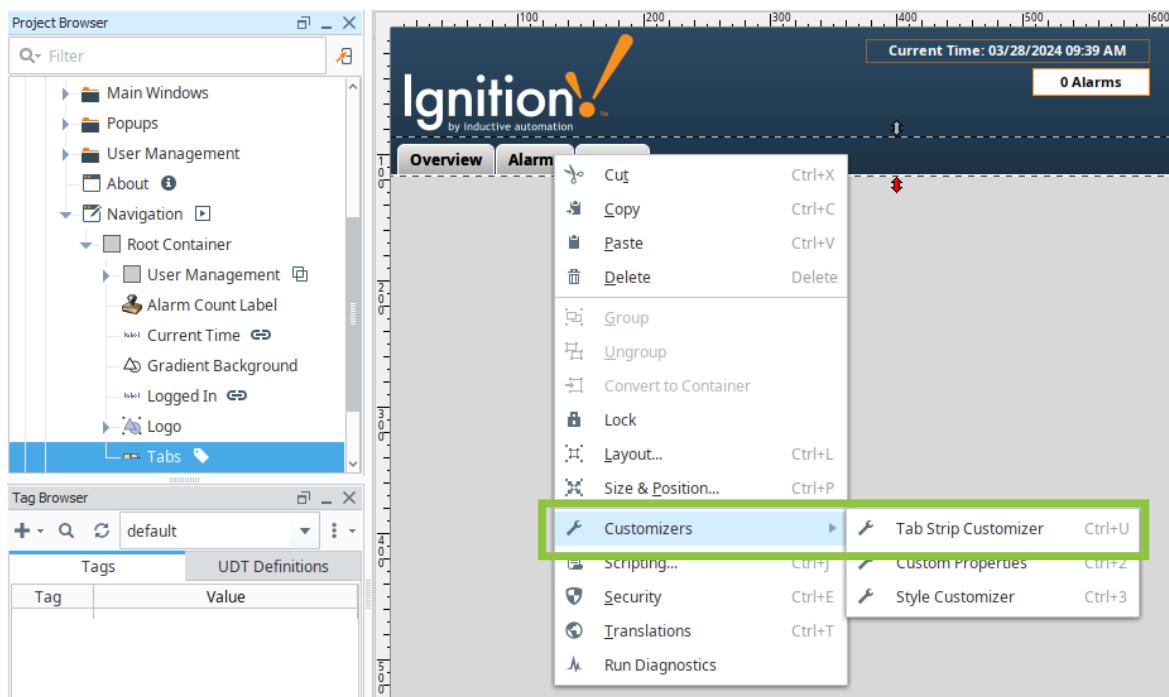
It is impossible to access a window in a Vision Client unless you provide some way to navigate to it. Popup windows are usually opened by another component; we will explore this in Chapter 18.

Main windows need to be added to some kind of navigation control. Vision includes a Tab Strip component to easily create navigation systems. However, since we based our project on the Vision Tab Nav project template, we already have a docked window that

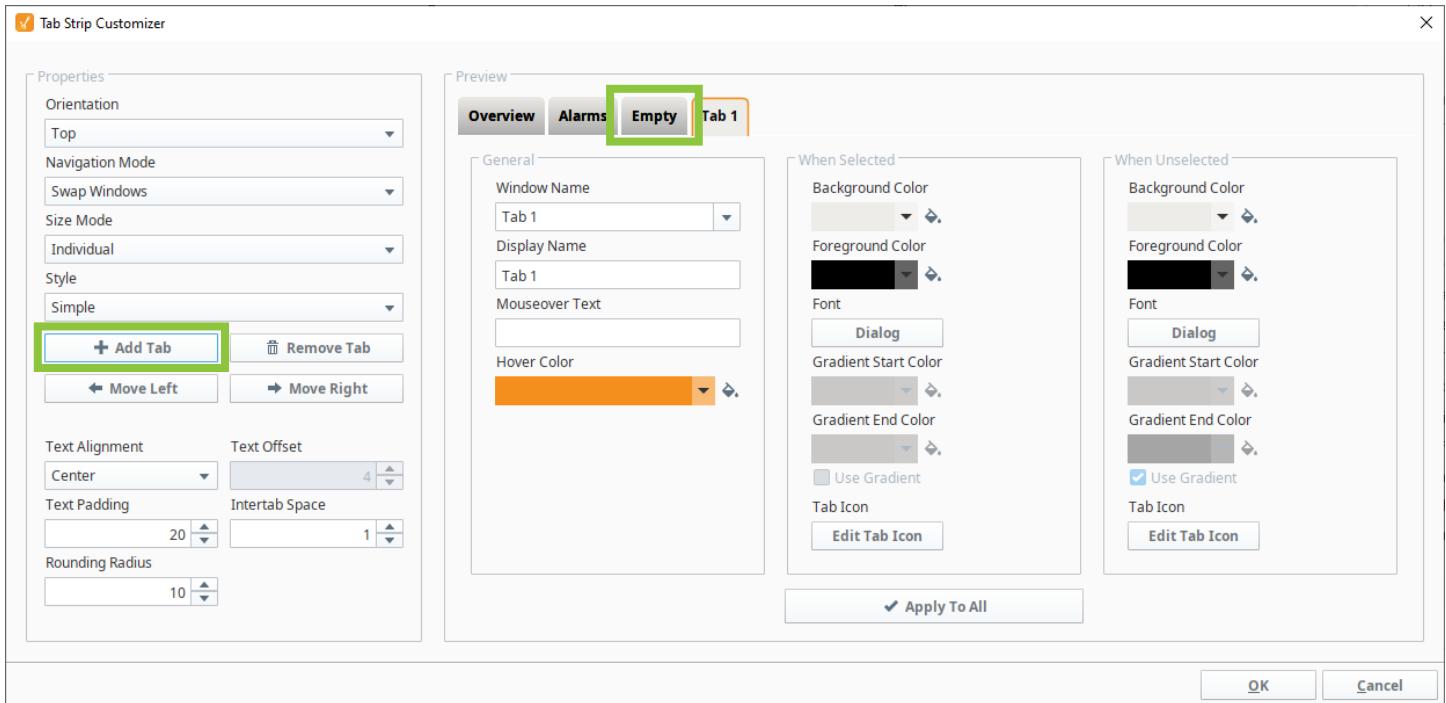
contains a Tab Strip component, so we will be able to simply add windows to this as we work through our project. There are two windows with a square and triangle icon to the right of them: Main Windows/Overview, and Navigation. These windows are set to open on startup so they are the first windows the user will see in the run-time environment.

To add a window to the navigation tab strip, follow these steps:

1. Double-click **Navigation** in the Project Browser.
2. Right-click the **Tab Strip** component.
3. Click **Customizers**.
4. Click **Tab Strip Customizer**.

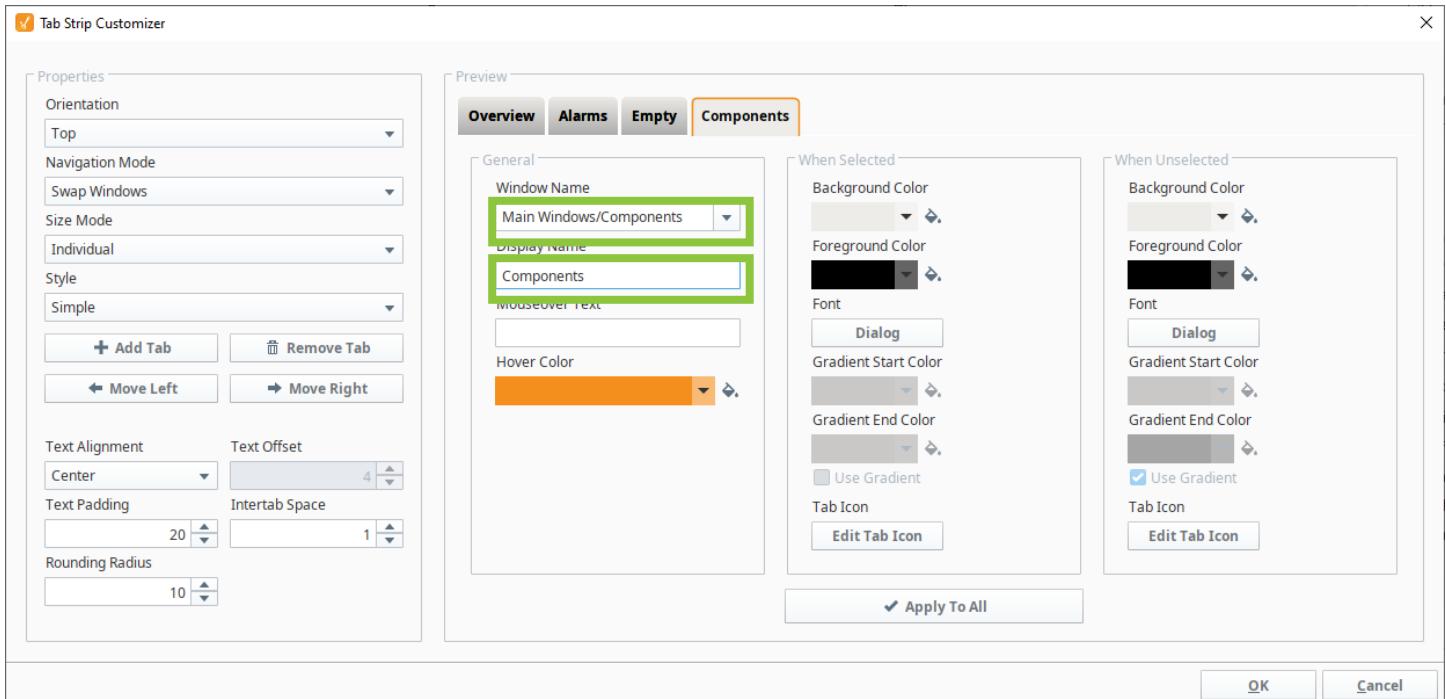


5. The Tab Strip Customizer window opens.
6. Click **Empty**.
7. Click **Add Tab**.



8. Select **Main Windows/Components** from the Window Name list.

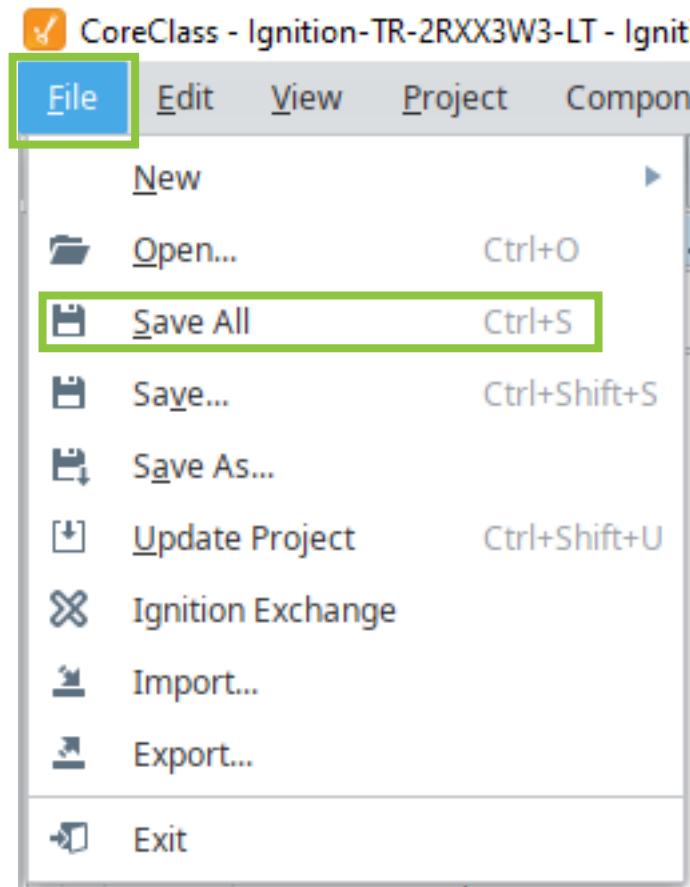
9. Enter **Components** in the Display Name text field.



10. Click **OK**.

The tab strip is updated to include the new window.

11. Click **File**.
12. Click **Save All**.

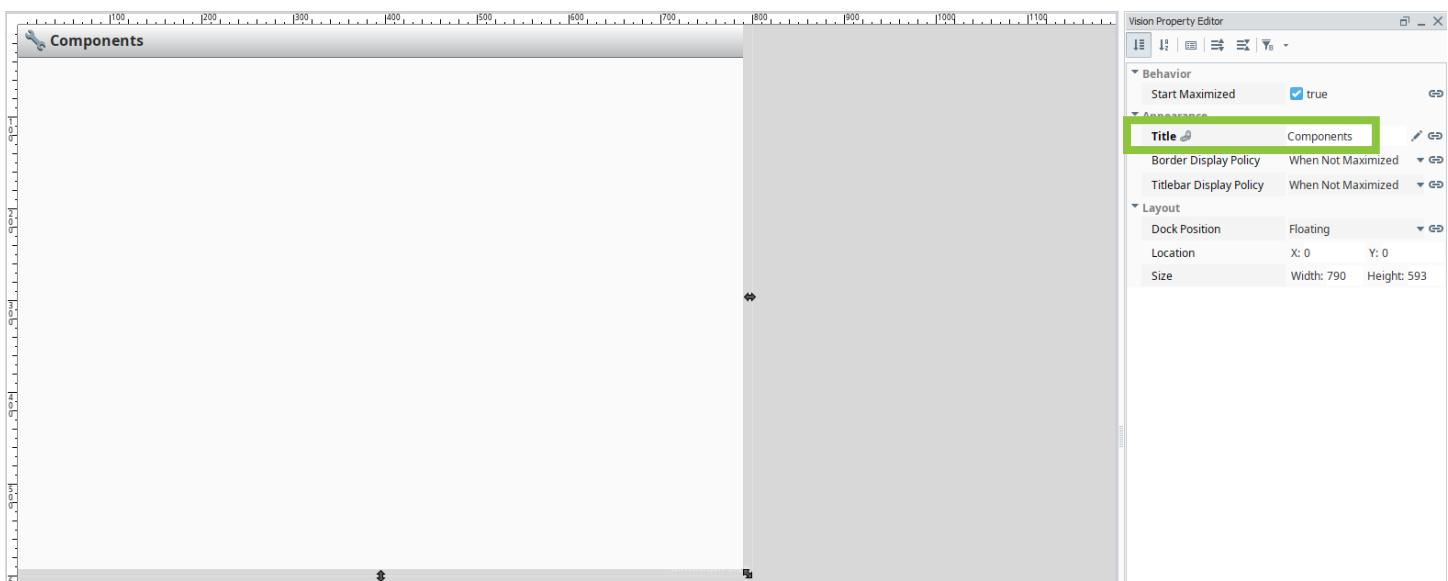


13. The project is saved.

Modifying Components on a Window

Now that we have a new window, we want to start adding components to it.

1. Double-click **Components** in the Project Browser.
2. Click the **Header Label** component.
3. Type **Components** in the Title property field on the Vision Property Editor.



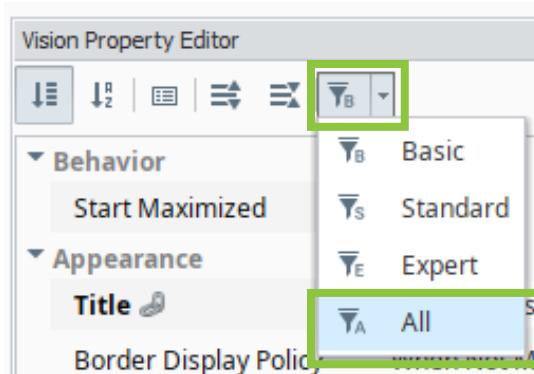
Changing the Header Icon

The header on the windows in the project template display a small icon next to the text. We would like to change that icon, but unfortunately, there does not appear to be a property to do that.

Showing All Properties

The property to change the header icon is not showing on the Vision Property Editor because by default, only a very small subset of properties displays. We need to change this behavior so that all properties of all components always display.

1. Click the small arrow to the right of **Filter Properties**.
2. Click **All**.

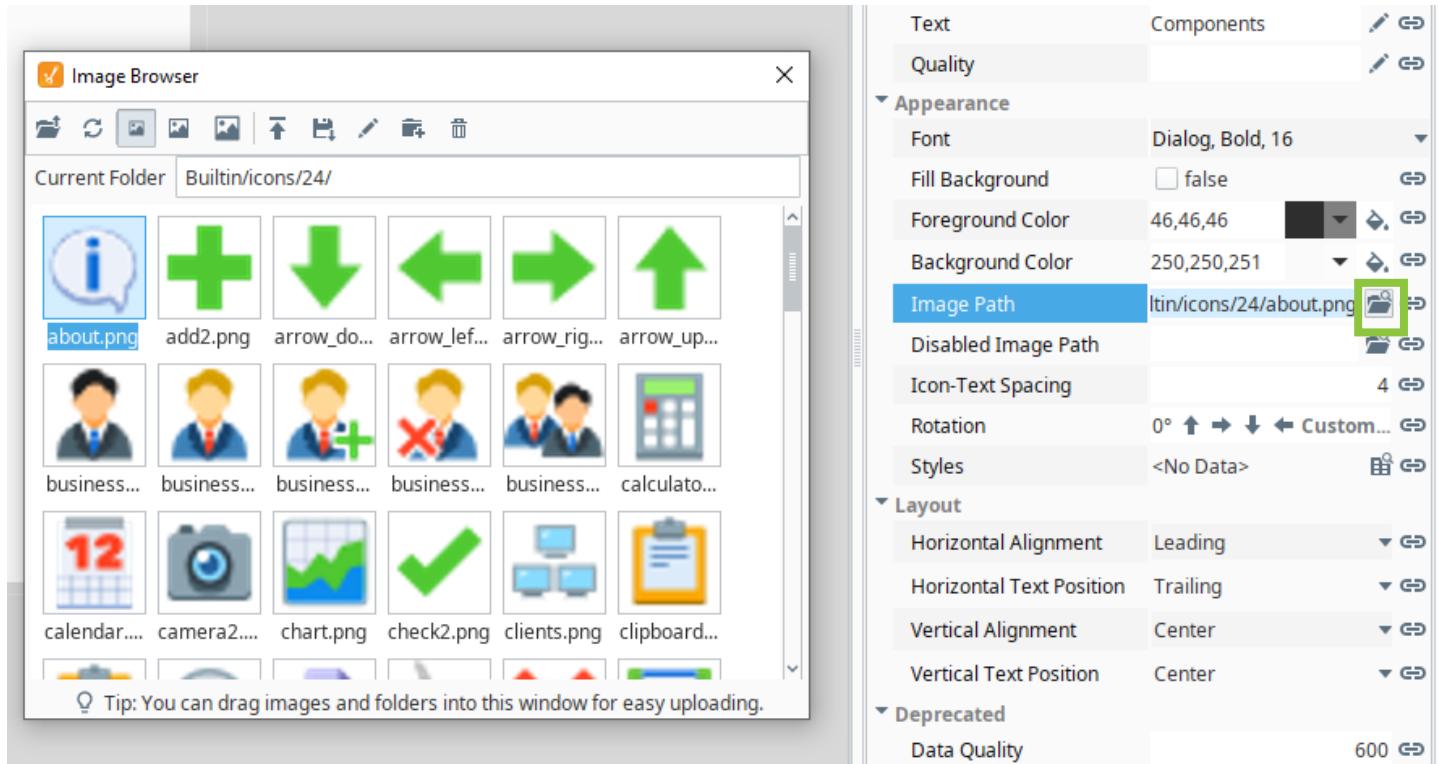


Thankfully, Designer will remember this setting, so you never need to worry about it again.

Changing the Icon

Now that we can see all properties, we can select a new icon.

1. Click the **folder icon** in the Image Path property.
2. Double-click any icon.

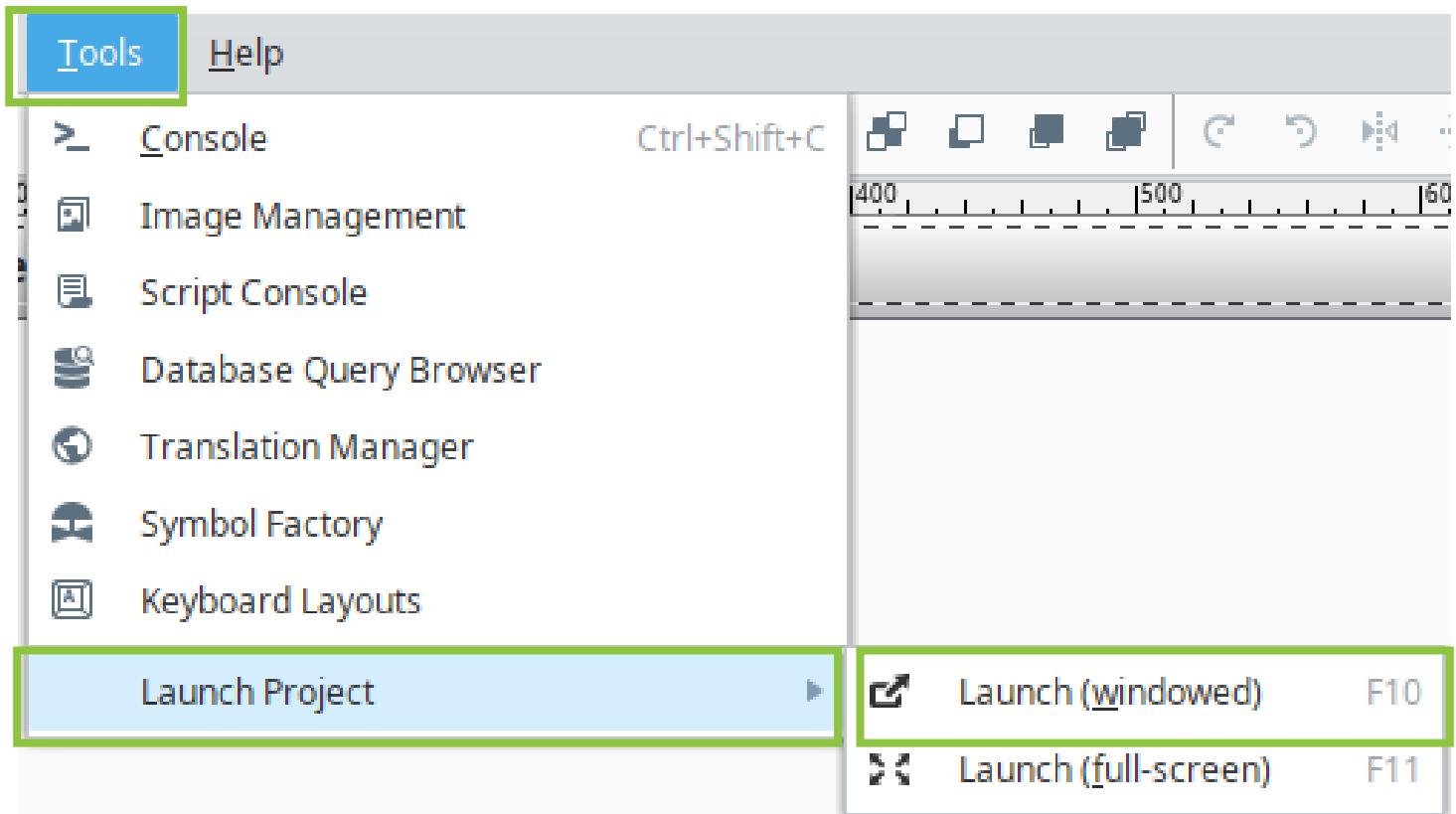


3. Click **File**.
4. Click **Save All**.

Previewing in the Client

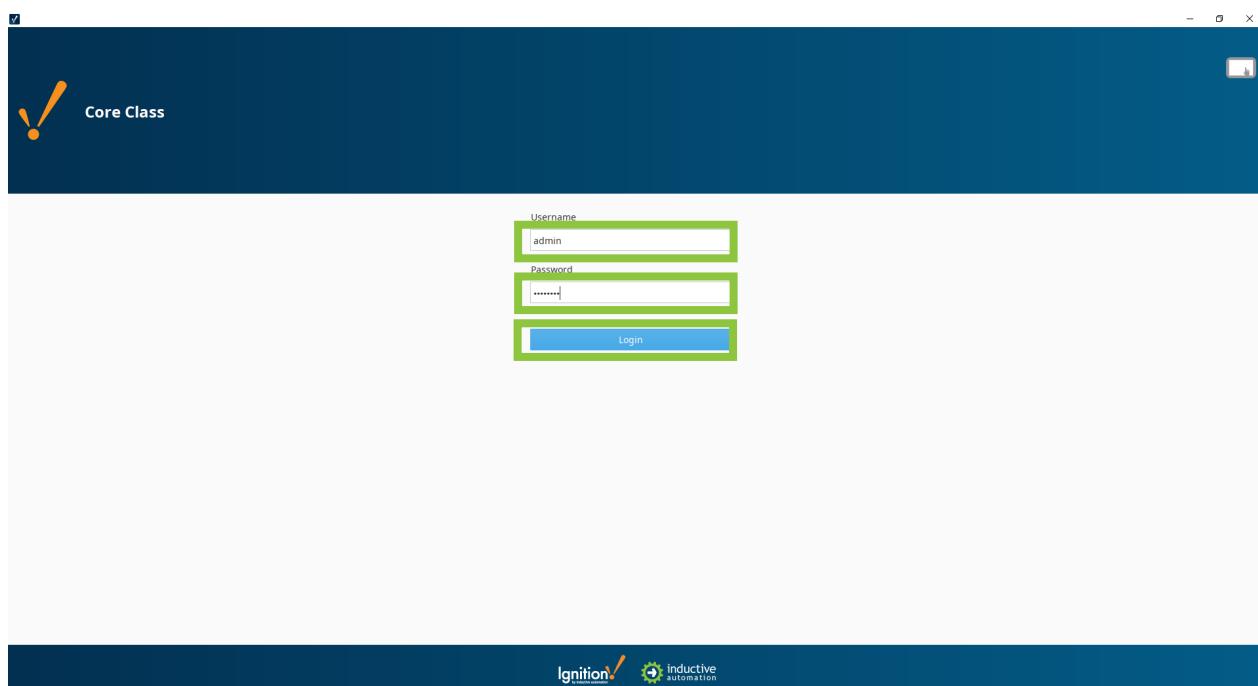
While the Designer provides us with a good idea of what our windows will look like, it's important to preview our project in the Vision client.

1. Click **Tools**.
2. Select **Launch Project**.
3. Click **Launch (windowed)**.



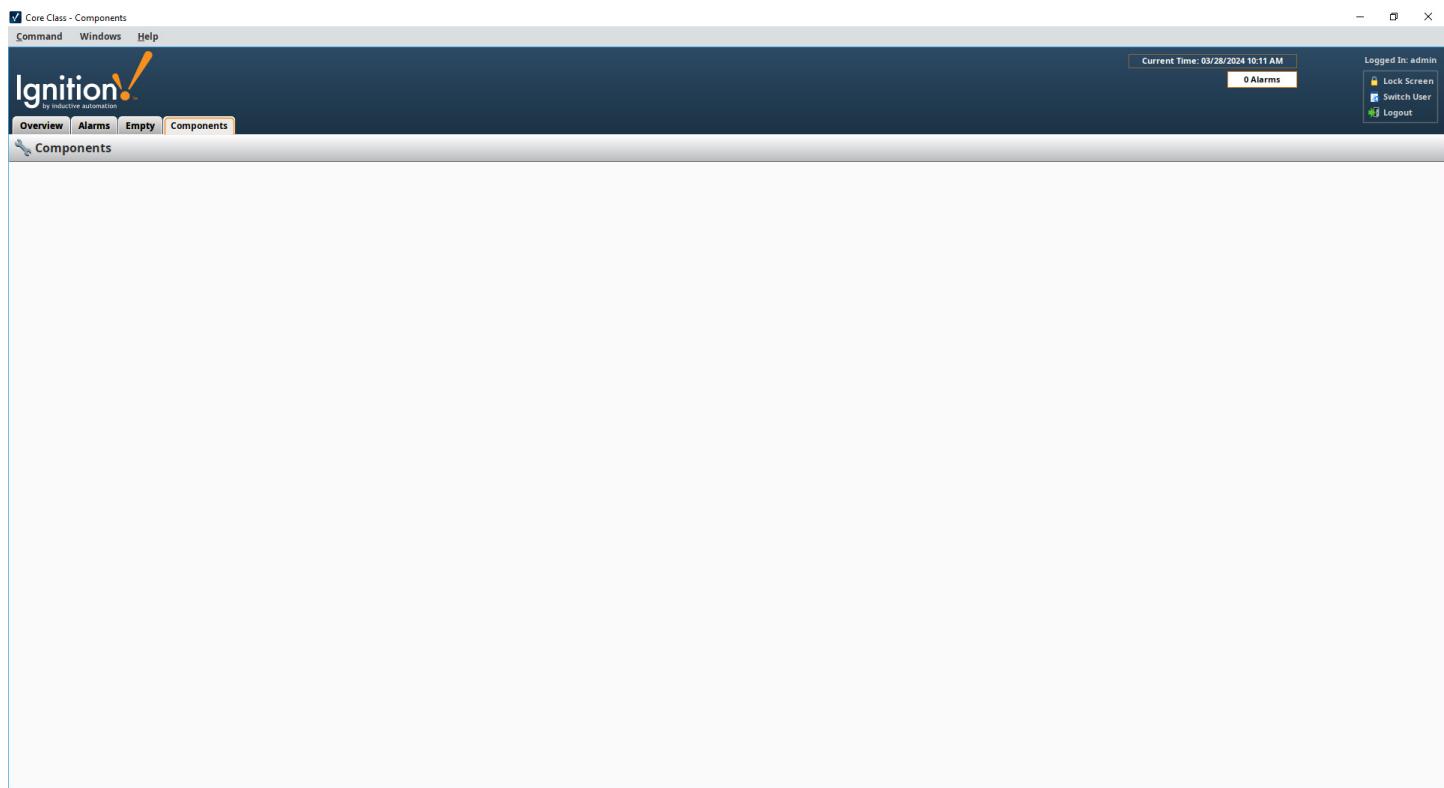
The Vision Client will launch.

4. Enter **admin** in the username text field.
5. Enter **password** in the password text field.
6. Click **Login**.



The client launches. The Overview window displays.

7. Click **Components**.



Launch Types

When you told Designer to open the client, there were two options.

Windowed mode is the usual client launch method. The client runs as its own fully independent application that can be moved or resized on your computer.

Full Screen mode also runs as a fully independent application, but the client will not have a title bar or borders, and covers the operating system task bar. If you accidentally launch in full screen mode, you can click Command, then Exit to close the client.

Downloading the Vision Client Launcher

While we will be opening the client from within Designer, it is worth noting that there is also a separate client launcher application available. In many cases, you will not want to give your operators access to Designer; they only need to launch and interact with the client.

Download and Install the Client Launcher

Follow these steps to download and install the client launcher:

1. Open the **Gateway web page**. If you need help, see “Accessing the Gateway” on page 29.
2. Click **Download Vision Client Launcher**.

Vision Client Launcher

Download the Native Client Launcher to open Vision clients from any Ignition Gateway.



[Download Vision Client Launcher](#)

3. Click **Download for Windows**.



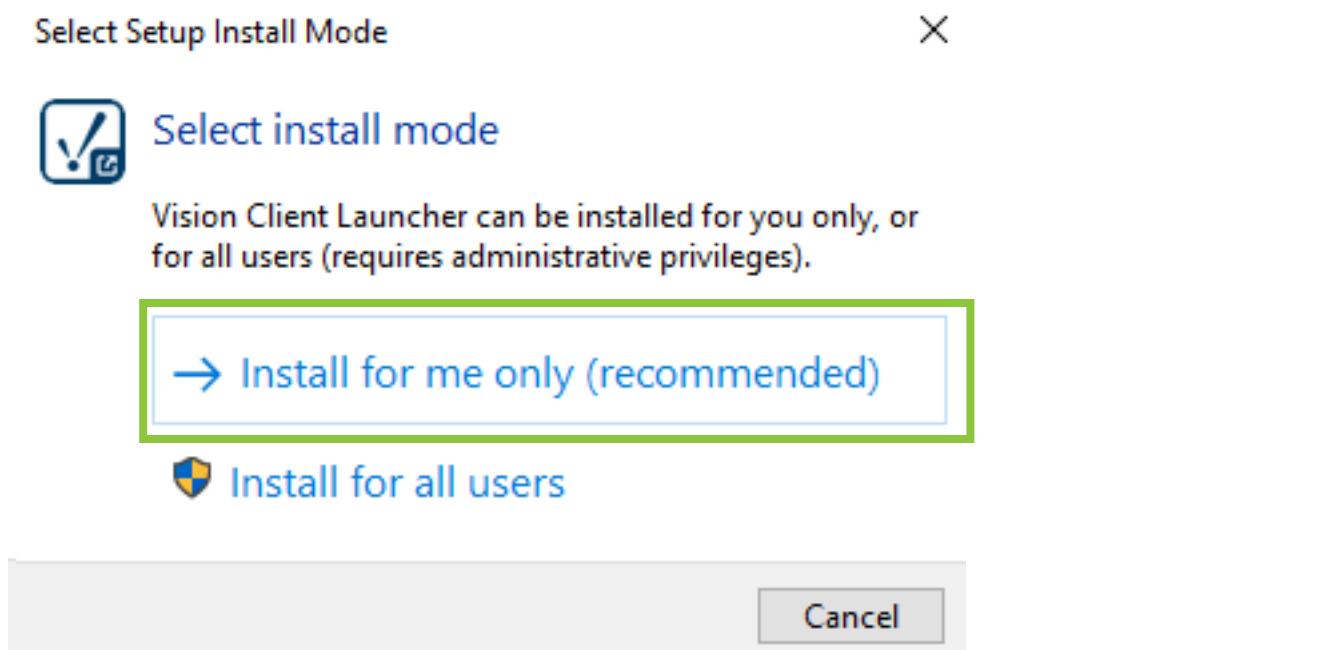
Download the Vision Client Launcher

[Download for Windows](#)

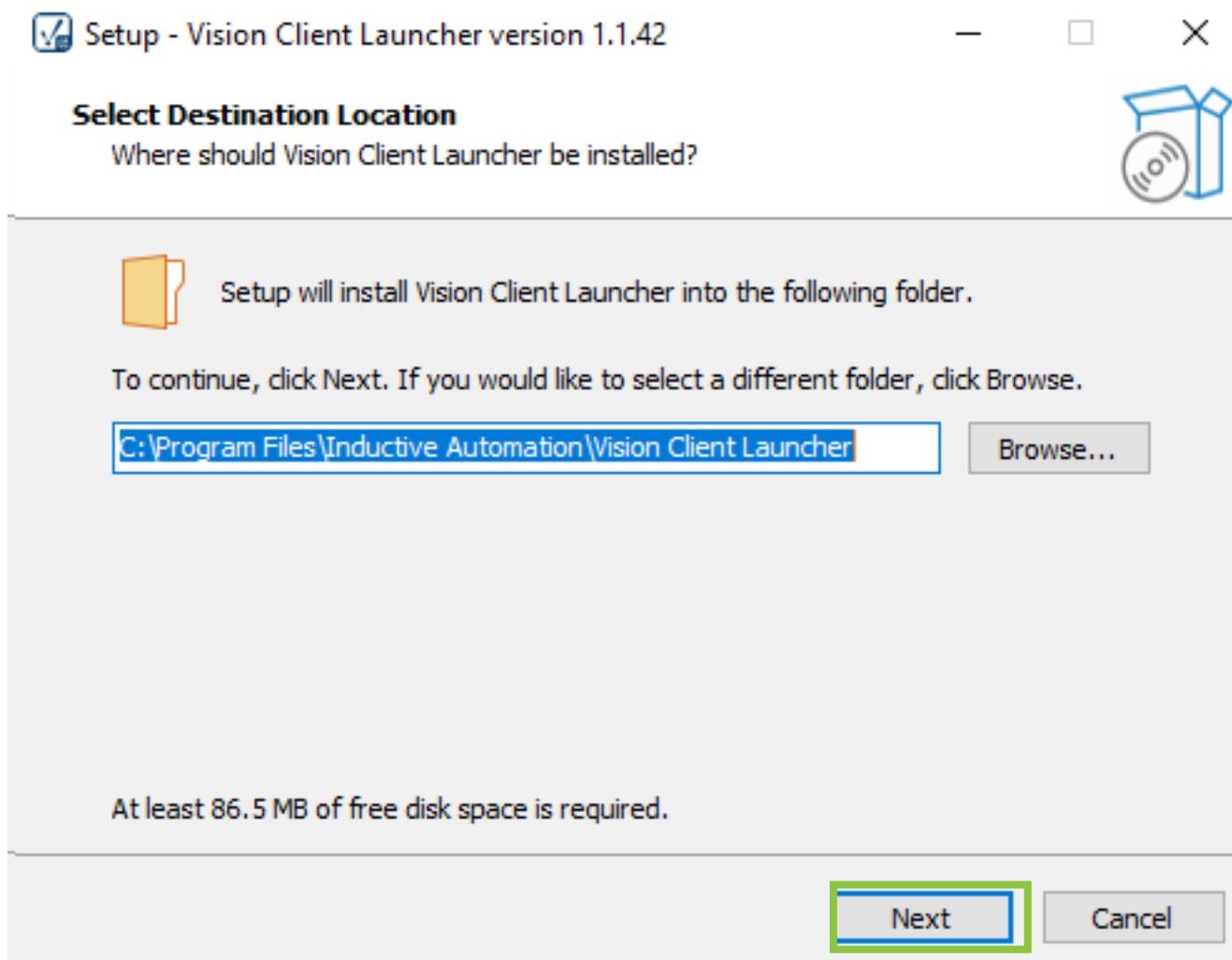
We've detected you're on Windows. Download the Vision Client Launcher for Windows and follow these steps below to install.

The installer downloads. It will most likely go into the **Downloads** folder in Windows.

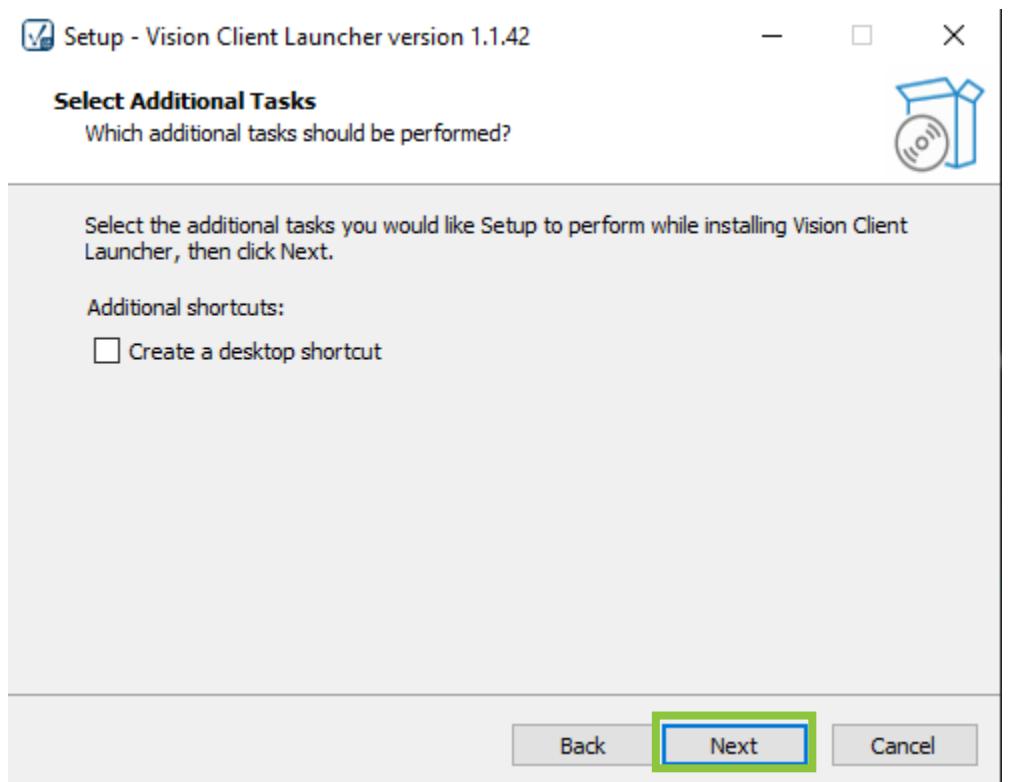
4. Double-click the **installer**.
5. Click an option to install for only you or for all users.



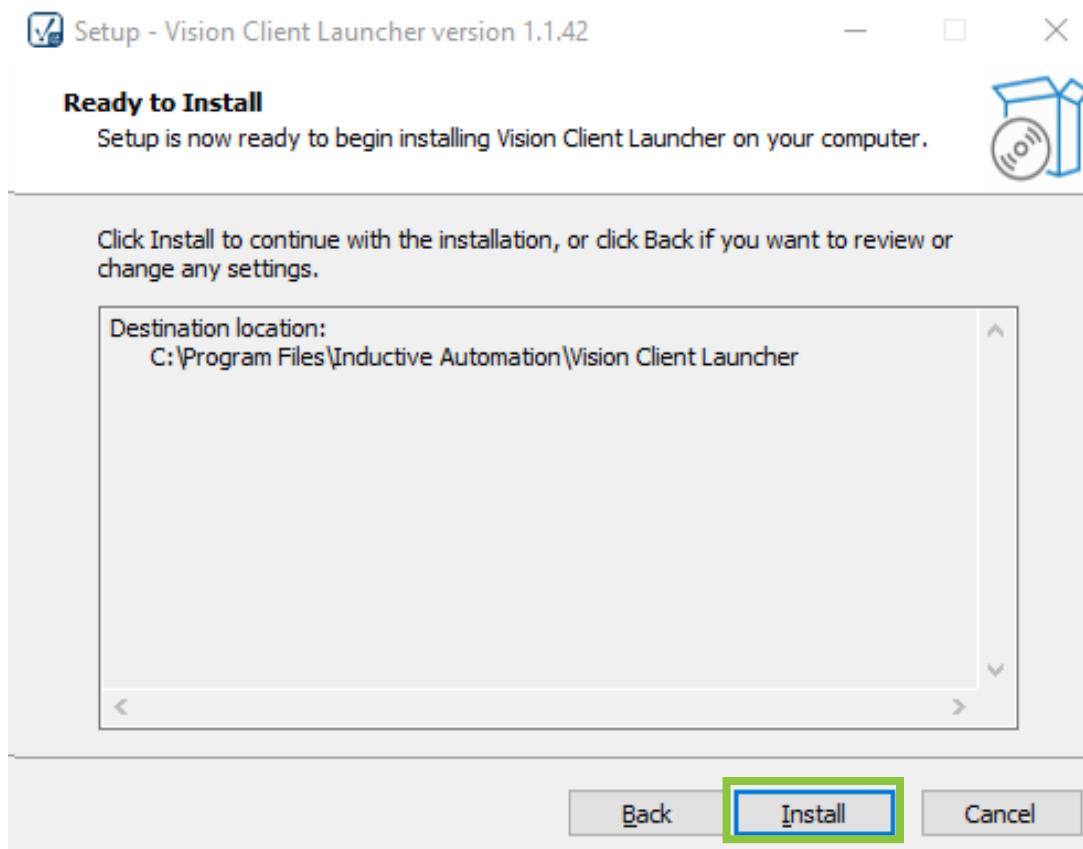
6. Click **Next**. We will install in the default location.



7. Click **Next**. We do not need a desktop shortcut.

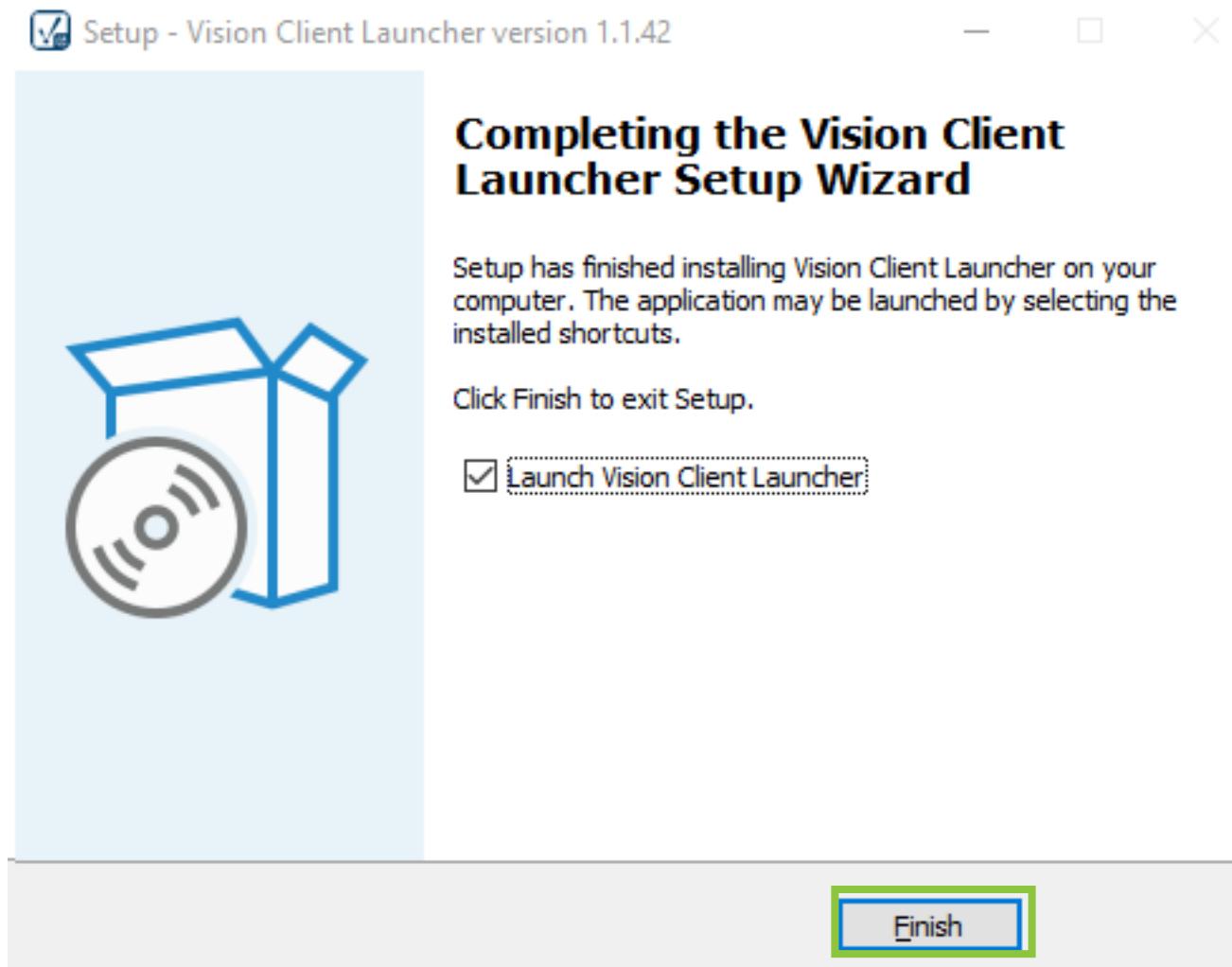


8. Click **Install**.



The Vision Client launcher will install.

9. Click **Finish**.

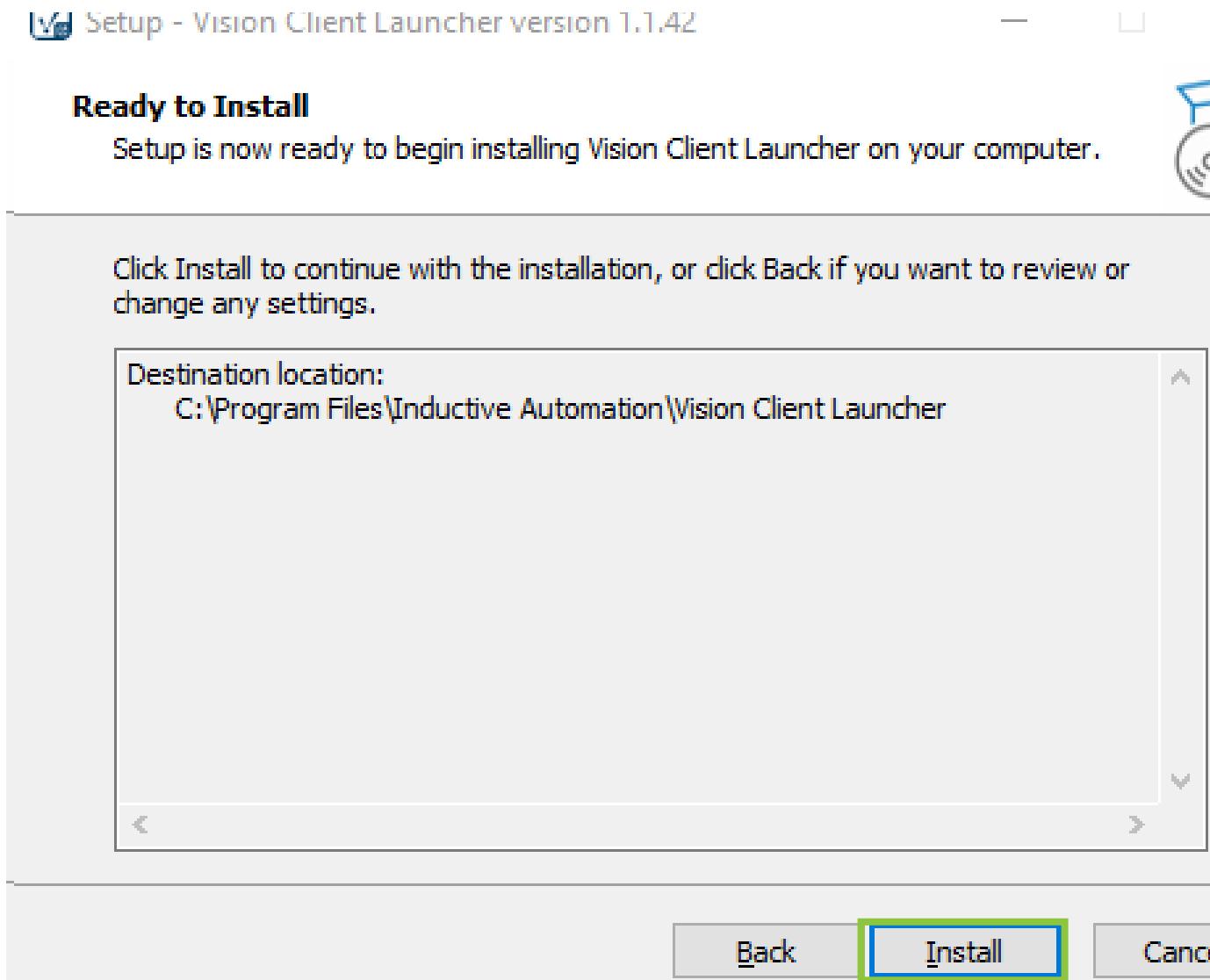


The launcher will start automatically.

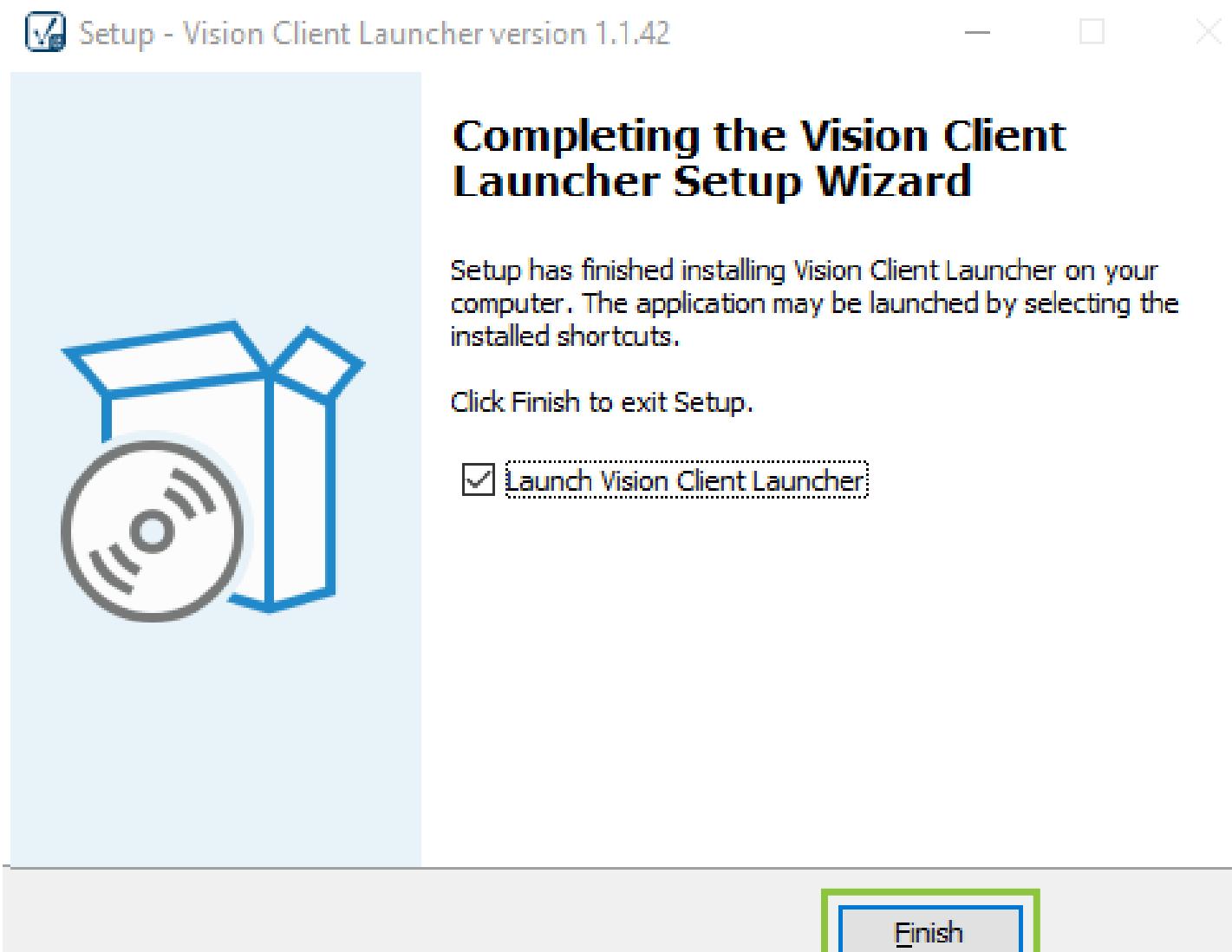
Connect to a Gateway

Similar to what we did in Designer, the first thing we need to do with the launcher is to connect it to a Gateway.

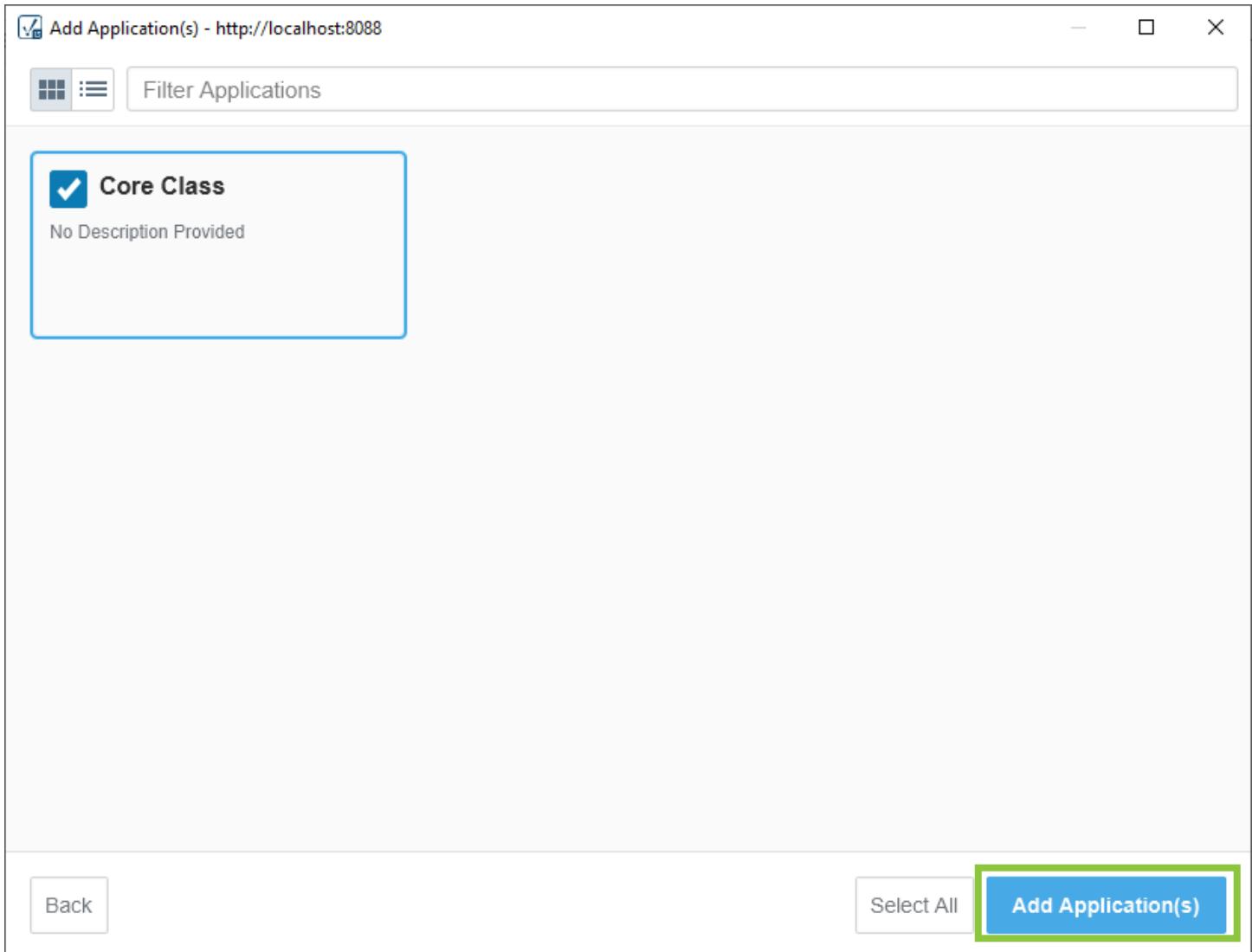
1. Click **Add Application**.



2. Click your **Gateway** from the list. It does not matter if you select it by host name or IP address.
3. Click **Select Gateway**.



4. Click **Core Class**.
5. Click **Add Application(s)**.



Now that the launcher is installed, the client can be launched from it at any time.

Vision Component Properties and Bindings

Components are the useful content of your windows. Most computer user interfaces contain components. They are the widgets you work with every day: the buttons, text areas, drop-downs, charts, and so on. The Vision module comes with a host of useful components out of the box, many of which are specialized for industrial controls use.

Configure Components

Configuring components is likely the bulk of a designer's work when designing a Vision project. The basic workflow is to take a component from the palette and drop it onto a window. From there, you can use the mouse to drag and resize the component, and you can use the Property Editor panel to change the component's properties, which changes the component's appearance and behavior.

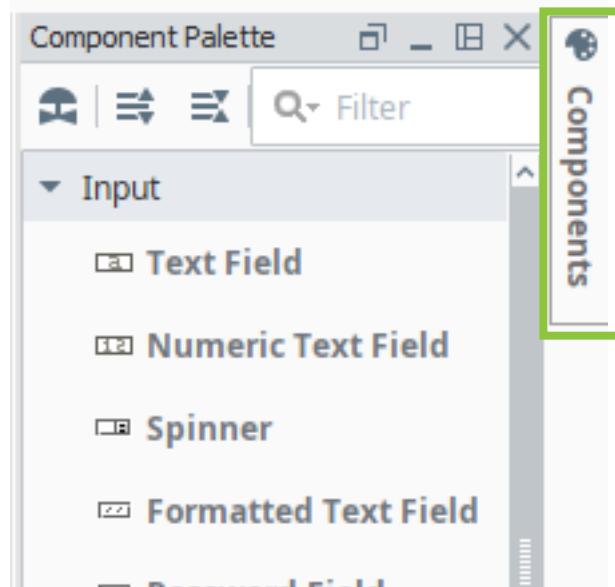
You can configure the following items for each component:

- Properties
- Custom Properties (Dynamic Properties)
- Size and Position
- Layout Mode
- Style Customizers
- Scripting
- Security

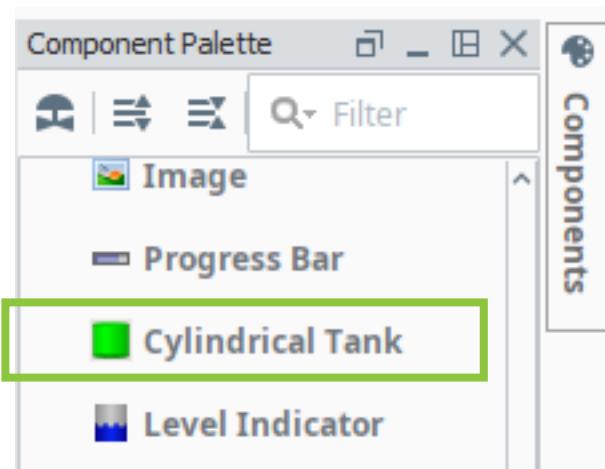
Add Components to Windows

You can add components to windows by dragging them from the Components Palette to the window.

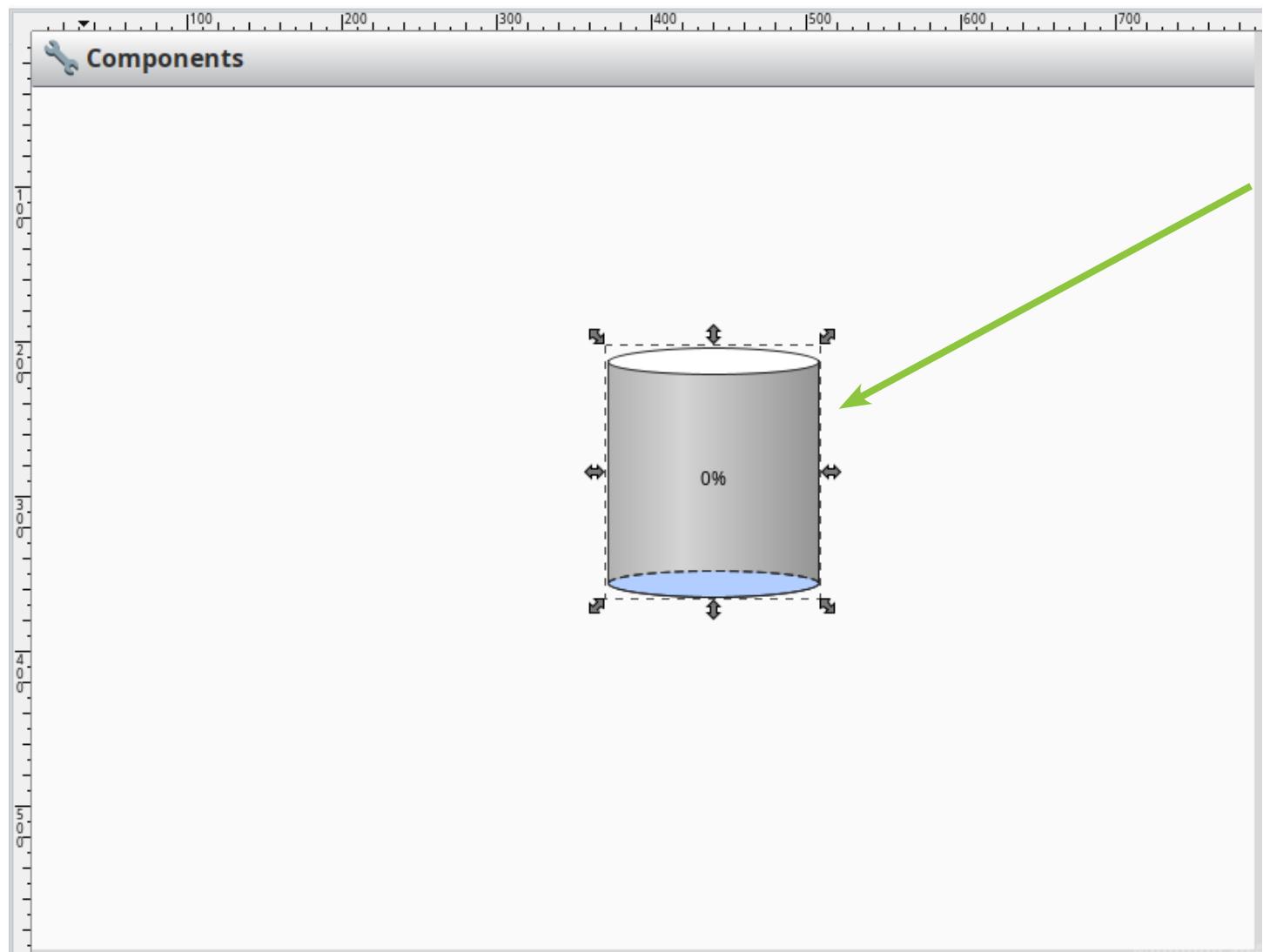
1. If necessary, open the **Components** main window.
2. Mouse over the **Components Palette** to expand it.



3. Press and hold your mouse button on **Cylindrical Tank**. You may need to scroll to see it.



4. Drag **Cylindrical Tank** onto the window. The precise location is not important.



Understand Component Layout

A component's layout settings determine its size and position. In many cases, we cannot know the exact size of the screen on which the client will eventually be displayed, so it is important to keep that in mind as we determine our layout options.

Vision clients have two options for layout: relative and absolute.

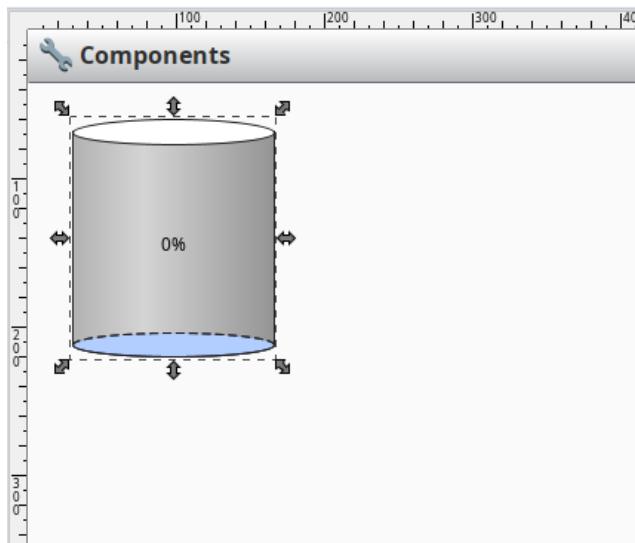
Relative Mode Relative layout, the default option, keeps a component's size and position relative to its parent container constant, even as the parent container grows or shrinks. More precisely, it remembers the component's position and size as a percentage of its parent's size. Relative mode also has the option of scaling a component's font appropriately. If the client ends up displaying on a larger screen than the one you used in Designer, the component will move or resize to maintain its relative dimensions,

Anchored Mode Anchored layout lets you anchor one or more of the component's edges to the container's edge. For example, if you anchor top and left, then your component stays a constant distance from top and left edges of its parent. If the client ends up being larger or smaller than the Designer window, the component will remain the exact distance from the anchored edges. Note that depending on how many sides the component is anchored to, this might still cause the component to resize.

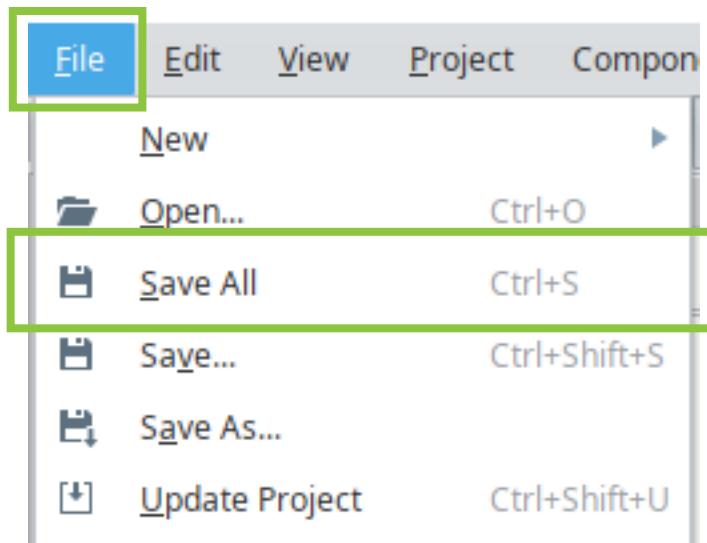
Explore Layout

To illustrate how component layout works, we are going to change the settings on the cylindrical tank we added earlier.

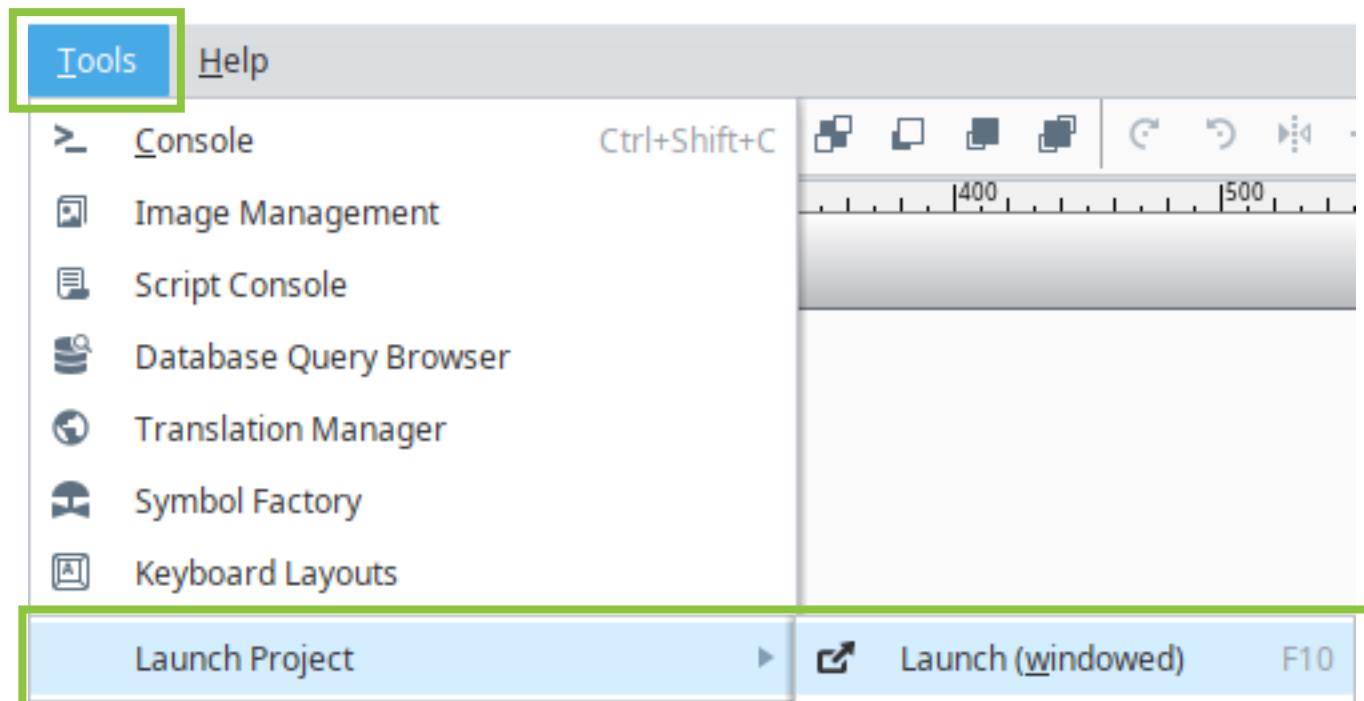
1. Click and drag **the tank** so that it is positioned near the top left corner of the window.



2. Click **File**.
3. Click **Save All**.



4. Click **Tools**.
5. Select **Launch Project**.
6. Click **Launch (Windowed)**.



The Client will open.

7. Enter **admin** in the username field.

8. Enter **password** in the password field.

Note: if you still have the client open from an earlier exercise, you can simply switch to it. You may still need to log in.

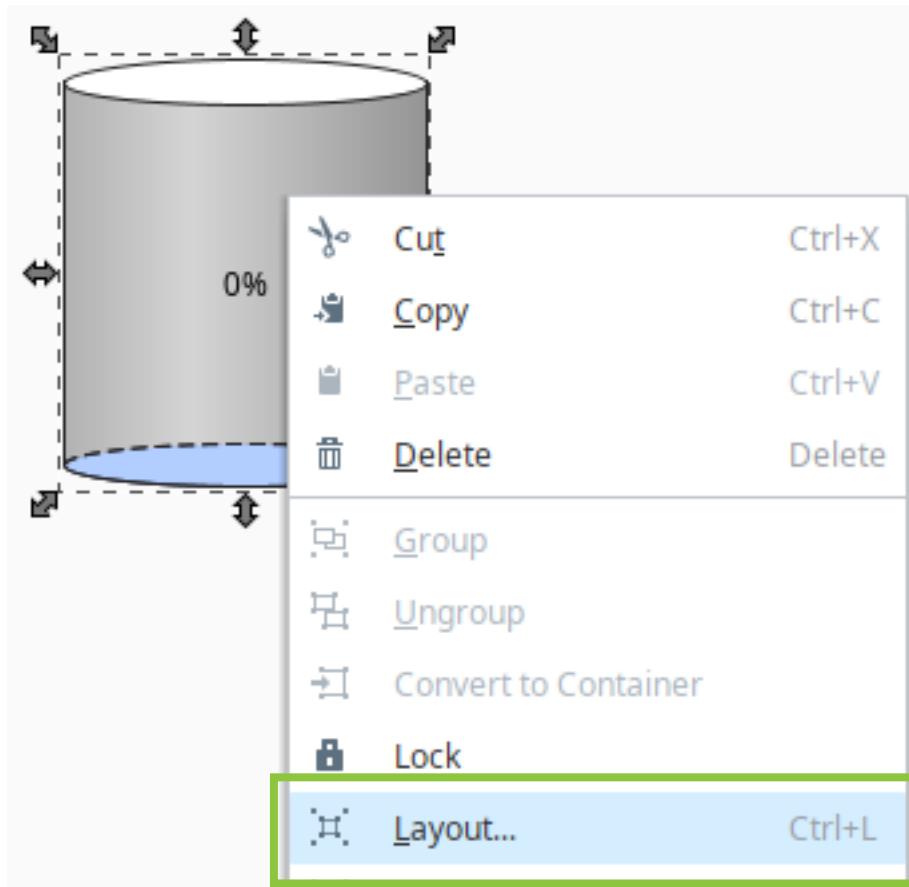
9. Click **Components**.

Because the tank is using the default layout mode (relative), it is larger in the client than it was in the Designer, and appears to be further from the edges of the window. In fact, it is proportionally both the same size and the same distance, percentage-wise, as it is in Designer.

10. Return to **Designer**.

11. Right-click the **tank**.

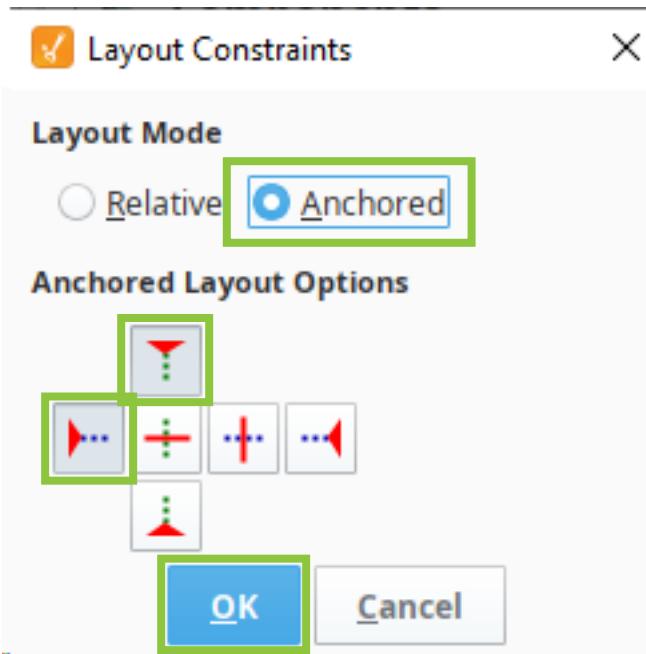
12. Click **Layout**.



13. Click **Anchored**.

14. Ensure that the **top** and **left** anchors are selected.

15. Click **OK**.



Note that the top and left sizing arrows on the tank are now red. This indicates that the tank is using anchored layout, and that those sides are anchored.

16. Click **File**.
17. Click **Save All**.
18. Switch back to the client.

You will see that the tank is now in the exact position, and exactly the size, it was in Designer.

There is no right or wrong to choosing to layout your windows using relative or anchored mode. Each has its advantages and disadvantages. Assessing the specific needs of each project is the only way to determine which mode to use.

Note: You can change the default layout mode by clicking Project, then Properties. Click Vision, then Design, and select the desired default settings.

Component Properties

Each component has a unique set of properties. A property is simply a named variable that affects something about the component's behavior or appearance.

Component properties can be set to specific values, or they can be bound to other components, to tags, or to other data sources.

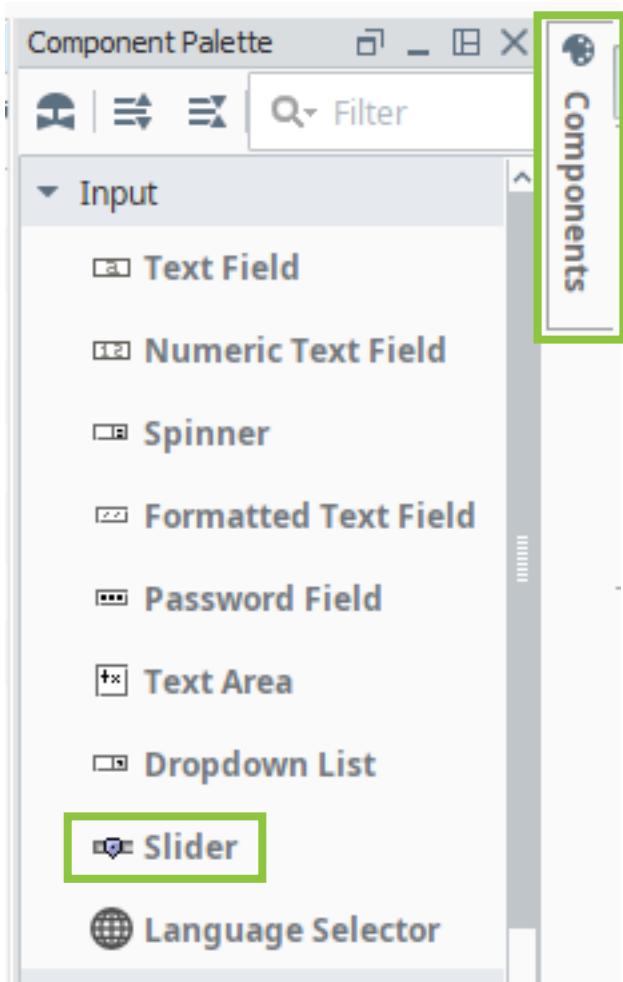
Bind Components to Other Components

Components can have their properties bound to other components' properties. This allows one component to directly control the properties of another component.

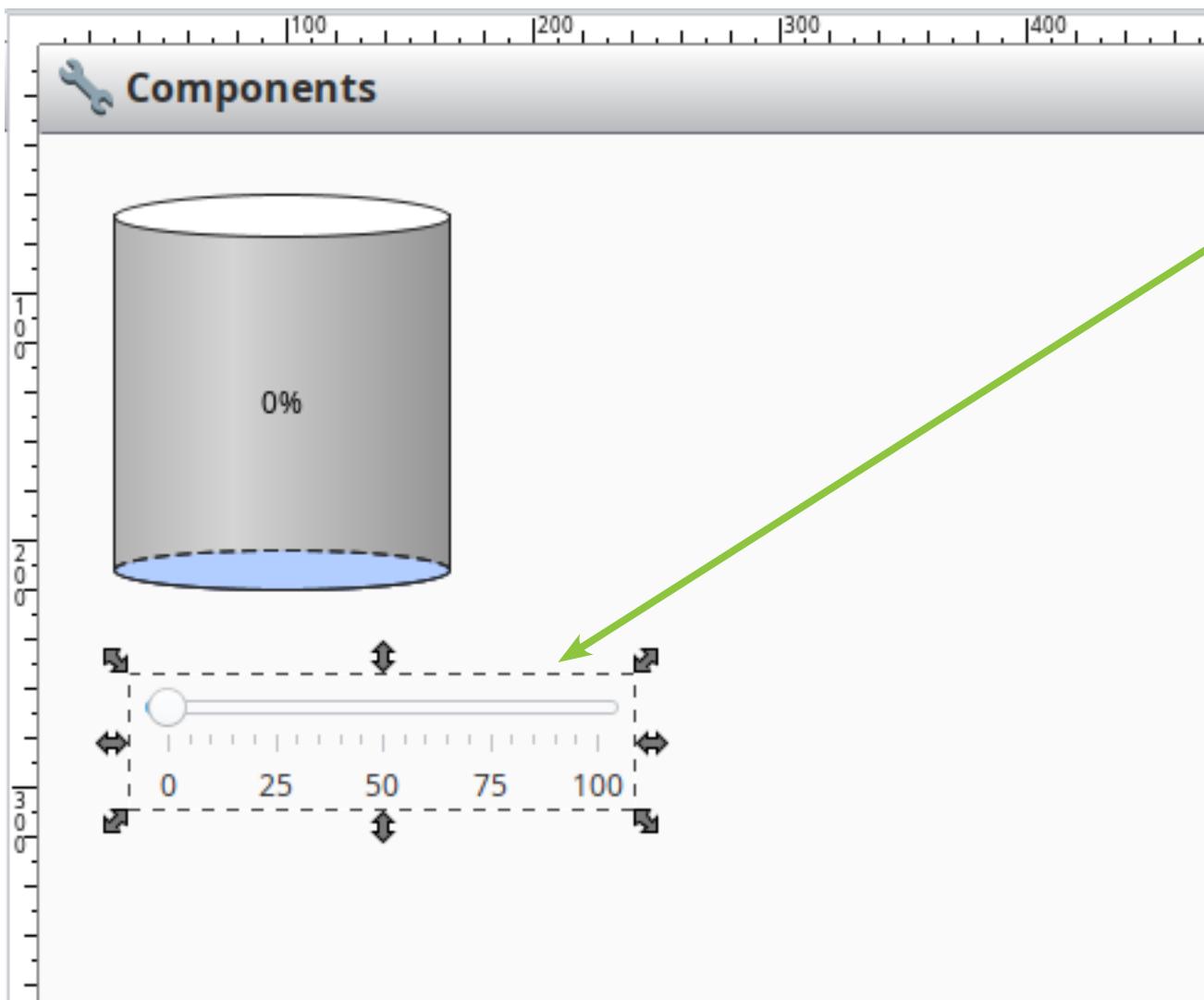
To see an example of a property binding, follow these steps:

1. Mouse over the **Components Palette** to expand it.
2. Scroll to find the **Slider** component.

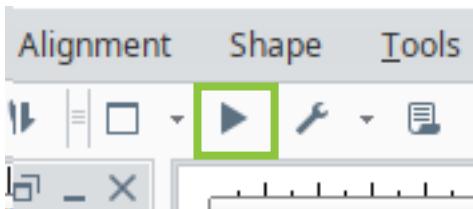
Note: You can use the filter text field at the top of the palette to type the name of the component you want to find.



3. Drag the **Slider** component to the window.

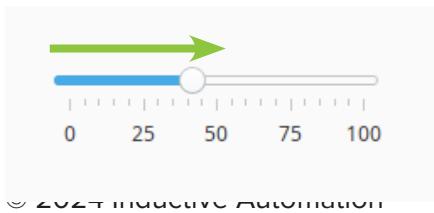


4. Click ► .

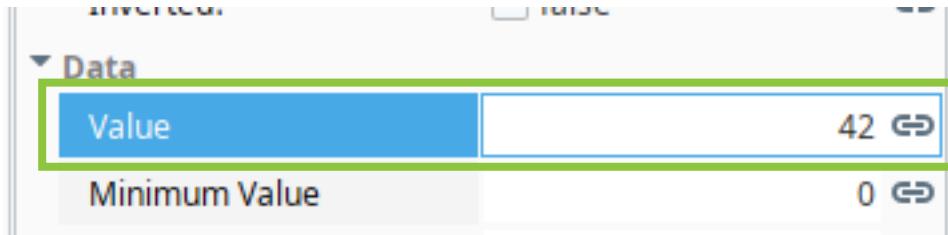


Preview mode allows you to interact with components, similar to how you can interact with them in the client.

5. Drag the **Slider's handle** to change its value.

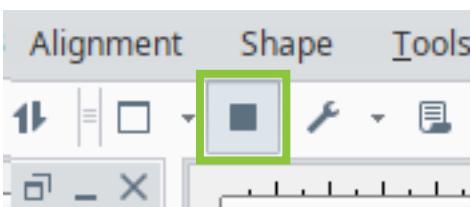


6. Look at the **Value** property on the Vision Property Editor.



The Value property represents the current position of the Slider's handle.

7. Click to exit preview mode.



8. Click the **Tank**.

The Tank also has a **Value** property. By changing its value directly on the Vision Property Editor, you can change how “full” the tank is.

As you can see, both the Slider and the Tank have a Value property. By binding the Tank’s value to the Slider’s, we can use the Slider to change the Tank.

9. Click the button to the right of the Tank’s **Value** property.



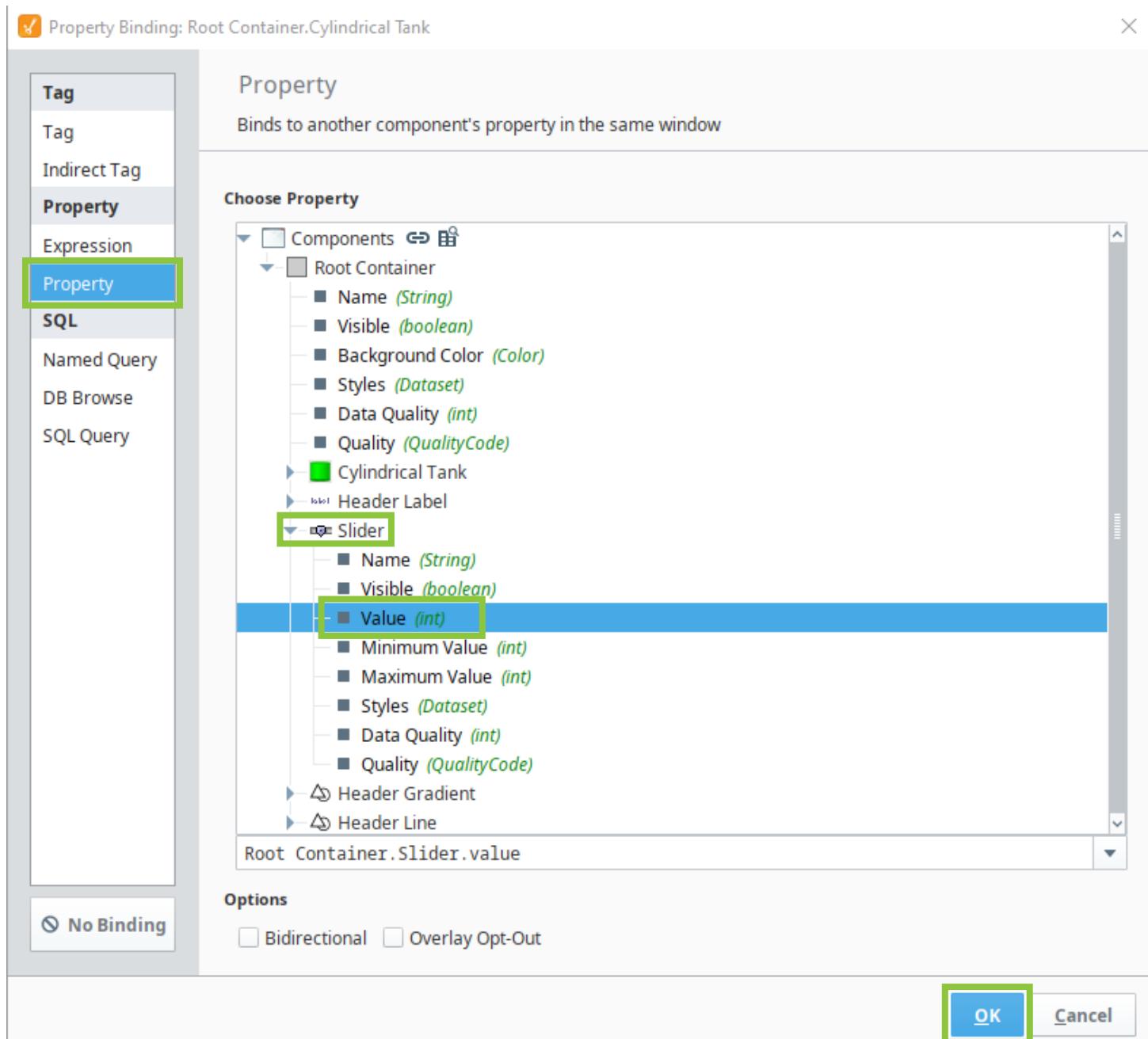
The **Property Binding** dialog box opens.

10. Click **Property**.

11. Expand **Slider**.

The properties of the Slider that can be used for bindings are listed.

12. Click **Value**.

13. Click **OK**.14. Click .15. Drag the **Slider's handle**.

The **Tank's value** changes to match the Slider's.

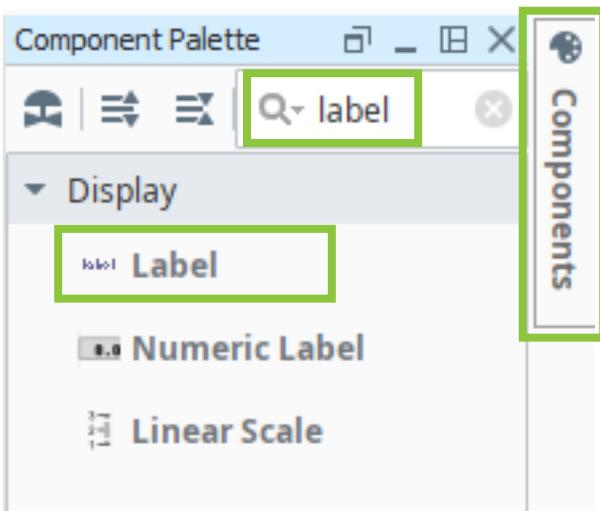
Note: You need to release your mouse on the Slider handle to change the Tank. The Slider does have a Defer Updates property that, when set to false, dynamically changes the Tank as you drag the Slider's button.

Binding Components Using Expressions

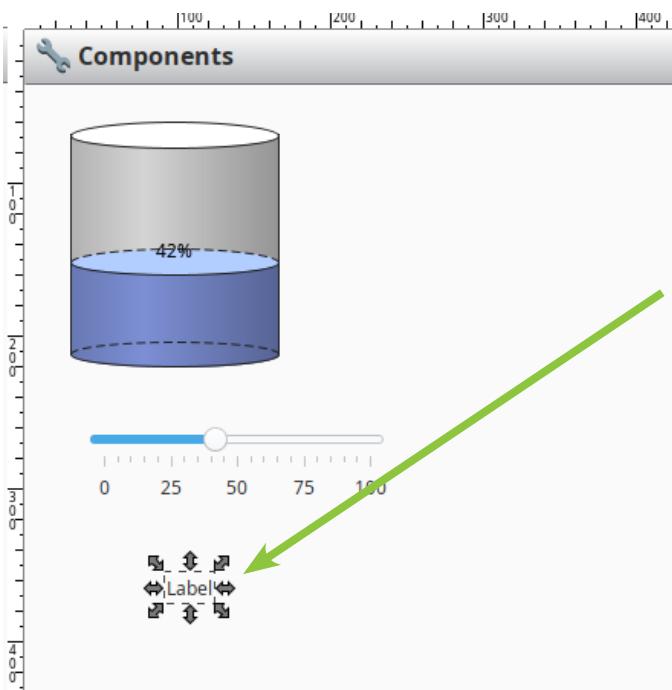
We can also set a binding that is more complex. The **Expression** binding will allow you to use more than one property in our expression language to create complex logic to return a single value. Let's add a Label to our example that displays the current value along with some descriptive text.

Add a Label

1. Expand the **Components Palette**.
2. Type **Label** in the filter box.



3. Drag the **Label** component onto the window.



You will see that Labels have a **Text** property. We can bind this to the Slider's **Value**.

We want to add an expression that combines the Slider's Value with literal text. Ignition's expression language is a custom-built simplified scripting language used to define dynamic values for component properties and expression tags.

If you have experience with programming languages, you will find the expression language very easy to use. If you do not have a programming background, don't worry—the expression language is easy to learn.

Everything in the expression language will evaluate to an expression; in other words, everything returns a value. Using the language is just a matter of figuring out what expression you need to write to return the value you want.

In our case, our end goal is when the Slider's **Value** is, for example, **42**, we want the label to say **42 gallons**. For that to work, we need to combine the Slider's Value with the literal text "gallons". The technical term for combining strings of text is concatenation, but you can think of it as simply adding the strings together, which is useful because we use the + sign for concatenation.

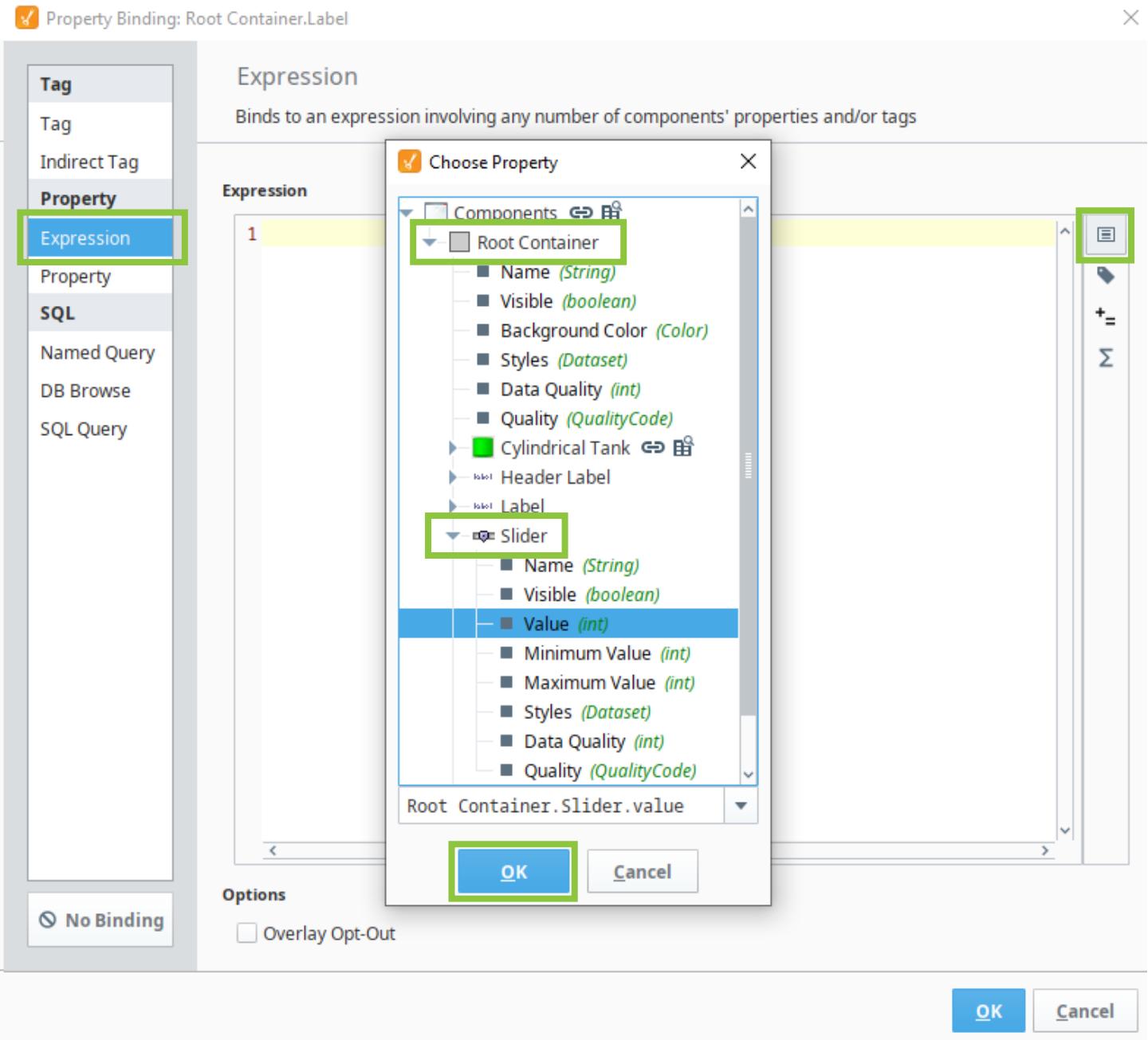
In order to reference a property of another component, we need to get its full path, and have it enclosed in curly braces. Fortunately, we never need to type that. The dialog that allows us to create bindings has buttons to allow you find component properties and enter them, so you'll always know that the path is correct.

Add an expression binding

1. Click  on the Label's Text property.



2. Click **Expression**.
3. Click .
4. Expand **Root Container**.
5. Expand **Slider**.
6. Click **Value**.
7. Click **OK**.



8. After the closing curly brace, enter + " gallons"

Note: Be sure to include the space inside the quotation marks.

9. Ensure your expression looks like this:

{Root Container.Slider.value} + " gallons"

10. Click **OK**.

You may need to resize the label to see all of the text.

11. Click .

12. Drag the **Slider's handle**.

Both the Tank's **Value** and the Label's **Text** change.

We will see other examples of expression bindings as we proceed through the class.

Use the Symbol Factory

The Symbol Factory is a resource included in Designer that contains hundreds of images to enhance your visualizations.

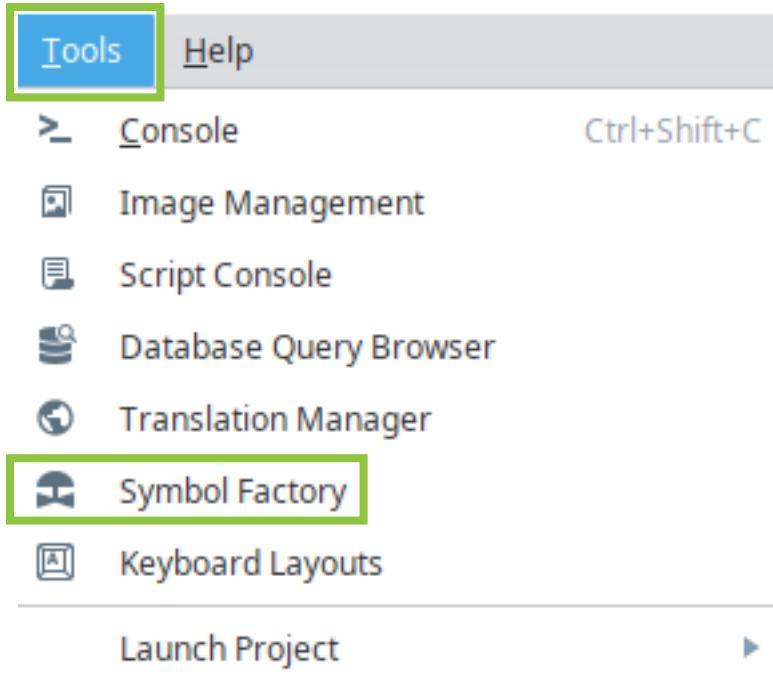
All images in the Symbol Factory were created as Scalable Vector Graphics (SVG), which means that they can be freely resized without impacting their quality.

The Symbol Factory contains many categories of images, grouped into two subsections: Basic and Enhanced. There are more images in the Basic subsection, but images in the Enhanced subsection allow for more editing possibilities.

Add a Symbol Factory Image

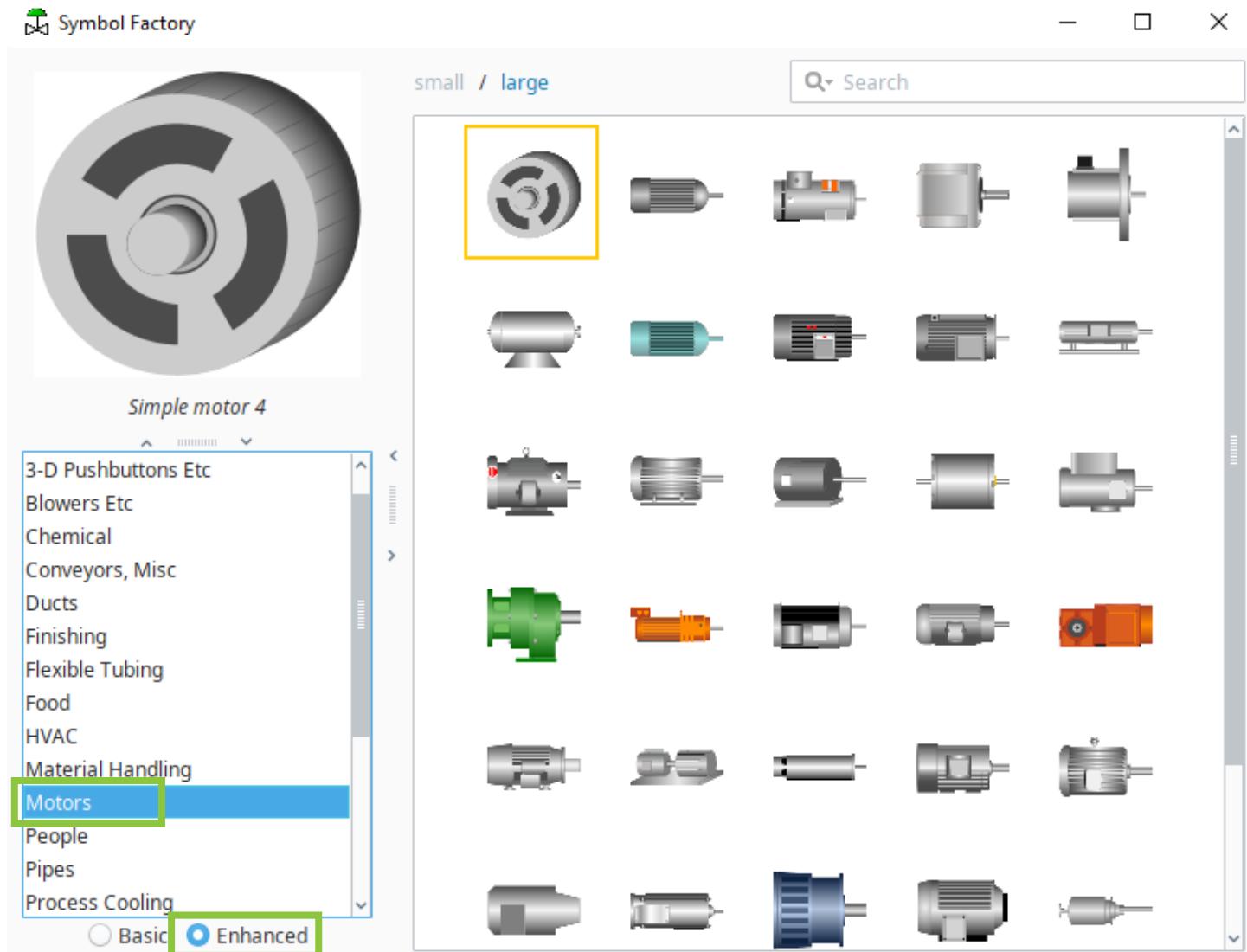
To add an image from the symbol factory, follow these steps:

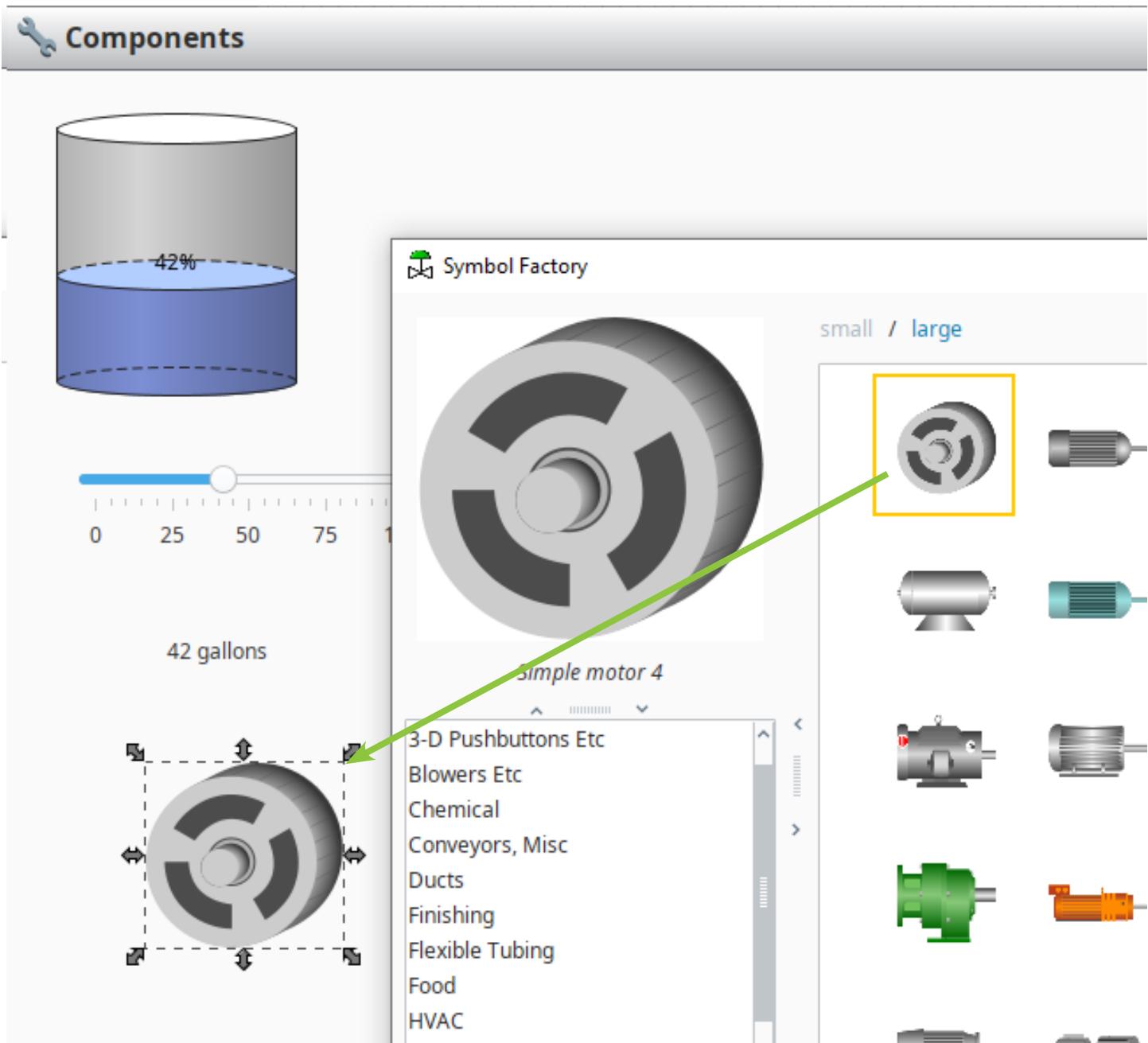
1. Click **Tools**.
2. Click **Symbol Factory**.



The Symbol Factory opens.

3. Click **Enhanced**.

4. Click **Motors**.5. Drag **Simple Motor 4** from the Symbol Factory to your window.



Note: You need to drag from the main grid of images. You cannot drag the large preview image.

6. Close the **Symbol Factory**.

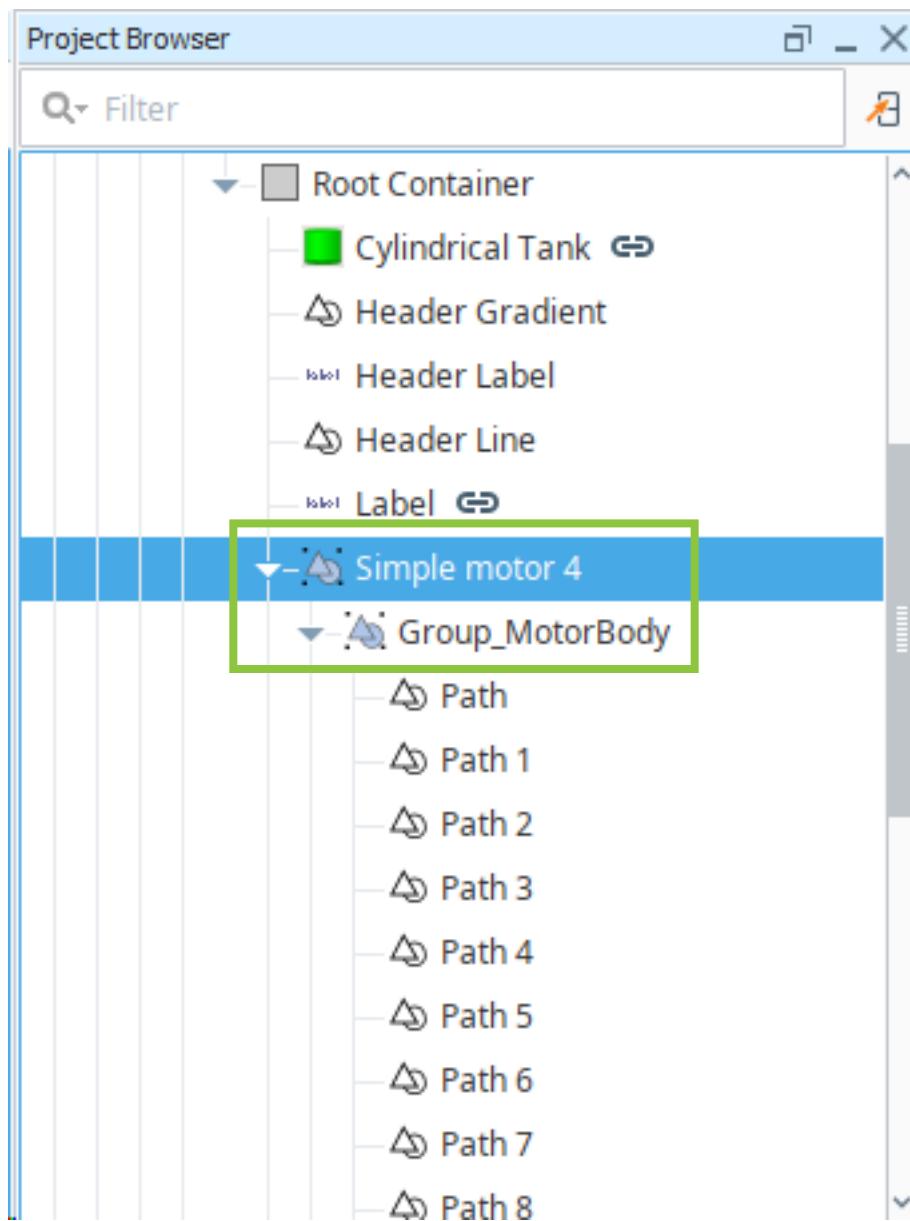
Explore a Symbol Factory Image

Symbol Factory images are really nothing more than collections of drawn shapes.

1. Click **Simple Motor 4**.

2. Expand **Simple Motor 4** in the Project Browser.

3. Expand **Group_MotorBody**.



The group expands to show the paths that make up the group.

In a later exercise, we will look at some of the possibilities of working with the groups that make up the shape, rather than the shape itself.

Explore the Vision Drawing Tools

Shapes such as lines, rectangles, and circles are created using the shape tools. Shapes are not components, but they have some similarities to them: each shape may be

individually selected, named, and assigned its own properties. But shapes have some additional capabilities that other components don't have, such as the ability to rotate.

By default, the shape toolbar is displayed on the right-hand edge of the Designer window, but you can drag it to wherever you prefer, just like the panels.

Shape tools allow you to create various shapes as well as edit them after they are created. Click on the tool's icon to make it the active tool, then click in the Designer and drag to place the tool in your workspace. You can also double-click on a shape to change to that shape's native tool. When a shape tool is active, a toolbar is displayed that has specific actions and settings for that tool.

After a shape is created, you can change its fill color, stroke color, and stroke style.

Selection Tool

The selection tool is active by default. In fact, we've been using it this whole time. When active, you can select shapes and components. Selected components can be moved, resized, and rotated.

Rectangle Tool

The rectangle tool creates and edits rectangle shapes. To create a rectangle, select the tool and drag inside a window, holding down Ctrl, makes a perfect square. Once a rectangle is created, you can use the square handles to change the rectangle's height and width.

There are also small circle handles that let you alter the rectangle's corner radius. Simply drag the circle down the side of the selected rectangle to make it a rounded rectangle. Hold down Ctrl to drag each rounding handle independently if you want non-symmetric corner rounding. You can use the Make Straight button in the rectangle tool's toolbar to return a rounded rectangle to be a standard straight-corner rectangle.

Ellipse Tool

The ellipse tool creates and edits circles and ellipses. You use it in much the same way as the rectangle tool. While it is the active tool, you can drag inside a window to create a new ellipse. Hold down Ctrl to make it a perfect circle. When an ellipse is selected, use the width and height handles to alter the shape.

Polygon Tool

The polygon tool creates polygons and stars. When this tool is active, you will see an additional toolbar at the top of the screen. You can use this toolbar to change the number of sides on the polygon, and to make it a star shape. Hold down Ctrl to keep the polygon's rotation an even multiple of 15°.

Arrow Tool

The arrow tool creates single or double-sided arrow shapes. When it is active, simply drag to create a new arrow. Use the check box on the toolbar to choose a single or double-sided arrow. To alter the arrow, use the diamond handles to change the two ends of the arrow, and the circle handles to change the size of the shaft and the head. When changing the arrow's direction, you can hold down Ctrl to snap the arrow to 15° increments.

Pencil Tool

The pencil tool draws freehand lines and shapes. When selected, you can draw directly on a window by holding down the mouse button. Release the mouse button to end the path. If you stop drawing inside the small square that is placed at the shape's origin, you create a closed path, otherwise you create an open path (line).

On the pencil tool's toolbar, there is a Simplify and Smooth setting, as well as a toggle between creating straight line segments or a curved line segments.

The Simplify option lets you specify a size in pixels that decreases the number of points used when creating the line. In general, points on the line are as far apart as this Simplify setting. If you find the line isn't accurate enough, decrease this setting. If you choose to create curved segments, the segments between points will be Bézier curves instead of straight lines. The Smooth option controls how curvy these segments are allowed to get.

Line Tool

The line tool draws lines, arbitrary polygons, or curved paths. Unlike all of the other tools, you don't drag to create new paths with the line tool. Instead, you click for each vertex you'd like to add to your path. To draw a straight line, simply click once where you want the line to start and double-click where you want the line to end. To make a multi-vertex path, click for each vertex and to end the line, double-click, press enter, or make a vertex inside the origin box.

As you draw the line, “locked-in” sections are drawn in green and the next segment is drawn in red. Hold down Ctrl at any time to snap the next segment to 15° increments.

On the line tool’s toolbar, you can choose between three modes: normal line, perpendicular, and curve. Perpendicular mode is just like line mode except that each segment is restricted to be either horizontal or vertical. Curve mode creates a Bézier curve path by attempting to draw a smooth curve between the previous two vertices and the new vertex.

Path Tool

The path tool edits all shapes and paths. This tool lets you directly modify the nodes in the path, add new nodes, remove nodes, and toggle segments between straight or curved.

Gradient Tool

The gradient tool affects the orientation and extent of any gradient paints.

Eyedropper Tool

The eyedropper tool sets the selected shape(s) and/or component(s) foreground/background or stroke/fill colors by pulling the colors from somewhere else in the window. When this tool is active, left-click to set the selection’s fill or background, and right-click to set the selection’s stroke or foreground. Note that this tool works on most components as well as shapes. For example, right-clicking sets the font color on a Button, or left-clicking sets the background color.

Setting a Shape’s Fill and Stroke

All shapes have three properties that affect how they look:

Fill Paint The interior color of the shape.

Stroke Paint The color of the shape’s outline.

Stroke Style The thickness, corners, and dash properties of the shape’s outline.

Editing Colors

The fill and stroke paints can be set to a variety of colors, using a variety of methods.

On the Fill Paint property, click on the **pencil icon** to open the color picker.

At the top of the panel, you can choose between the five different paint types:

No paint When used as a fill paint, it makes the interior of the shape transparent. Used as the stroke paint, does not draw the shape's outline.

Solid color A color that will completely fill the interior of the shape or the stroke. You can also set a transparency (alpha) level.

Linear gradient Smoothly blend any number of colors along a straight line across the shape. Each color is called a Stop. Each stop is represented as a dragable control in the gradient editor. You can click on a stop to select it and change its color or drag it to reposition it. You can right-click on it to remove it. You can right-click on the preview strip to add new stops and change the gradient's cycle mode.

Radial gradient Very similar to linear gradient except the colors emanate from a point creating an ellipse of each hue. Radial paints are configured in the same way as linear paint.

Pattern paint Uses a repeating pixel-pattern with two different colors. You can pick a pattern from the drop-down or create your own using the built-in pattern editor.

Stroke Style

The stroke style primarily controls the thickness of the line drawn, but you can also use it to create a dashed line.

The setting for thickness is specified in pixels. Dashed line can be applied by choosing the style from the list.

The effect of thickness and dash pattern settings is fairly self-explanatory, but the other stroke settings are a bit more subtle. You can notice their effect more easily on thick lines.

Tags

In Ignition, a Tag is more than simply what is considered a “Tag” in other systems. Each Ignition Tag starts as one data point, and may have a static or dynamic value that comes from a PLC device, an expression, or a SQL query. But once we have access to that data point, we can add configurations for alarming, scaling, historical storage, and more.

Ignition Tags are stored in Providers, and you can have as many providers as you want. By default, a fresh Ignition installation will have a single internal Tag provider.

Main Benefits of Tags

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

Drag-and-Drop Screen Design

Using the Vision and Perspective modules, you can drag-and-drop Tags to a Window or View to automatically create new components bound to the Tag. You can also bind Tags to existing components or properties by dragging the Tag onto the component.

Object-oriented Design

Tag UDTs (User Defined Types) can be used to design re-usable, parameterized, and extendable data types. You can create and configure new instances of these types in seconds.

Performance and Scalability

Tags offer 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 the Client, Tags improve efficiency with their lightweight subscription architecture. Adding additional clients creates a negligible effect on Gateway performance.

Integrated Component Feedback

Tags offer a quality and overlay system for Vision and Perspective 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.

Tag Values

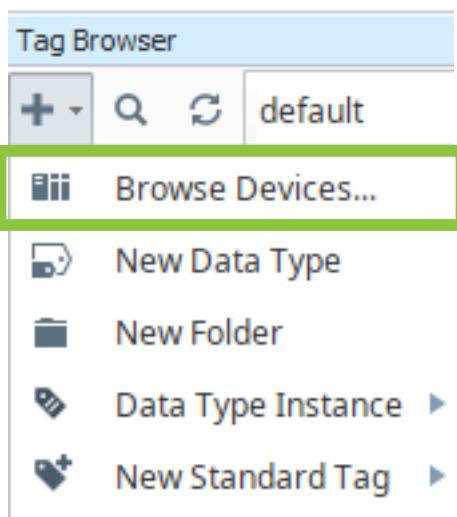
Fundamentally, the primary goal of a Tag is to represent a value in the Ignition system. The values can come from PLCs, expressions, queries, and so on, and can be used on Vision Windows, Perspective Views, Transaction Groups, and more.

Additionally, Tags offer alarming, scaling, meta data, and history. Depending on the specific type of Tag, even more options can be available. In general, Tags provide a common interface for tying together many types of data in Ignition.

Add Tags

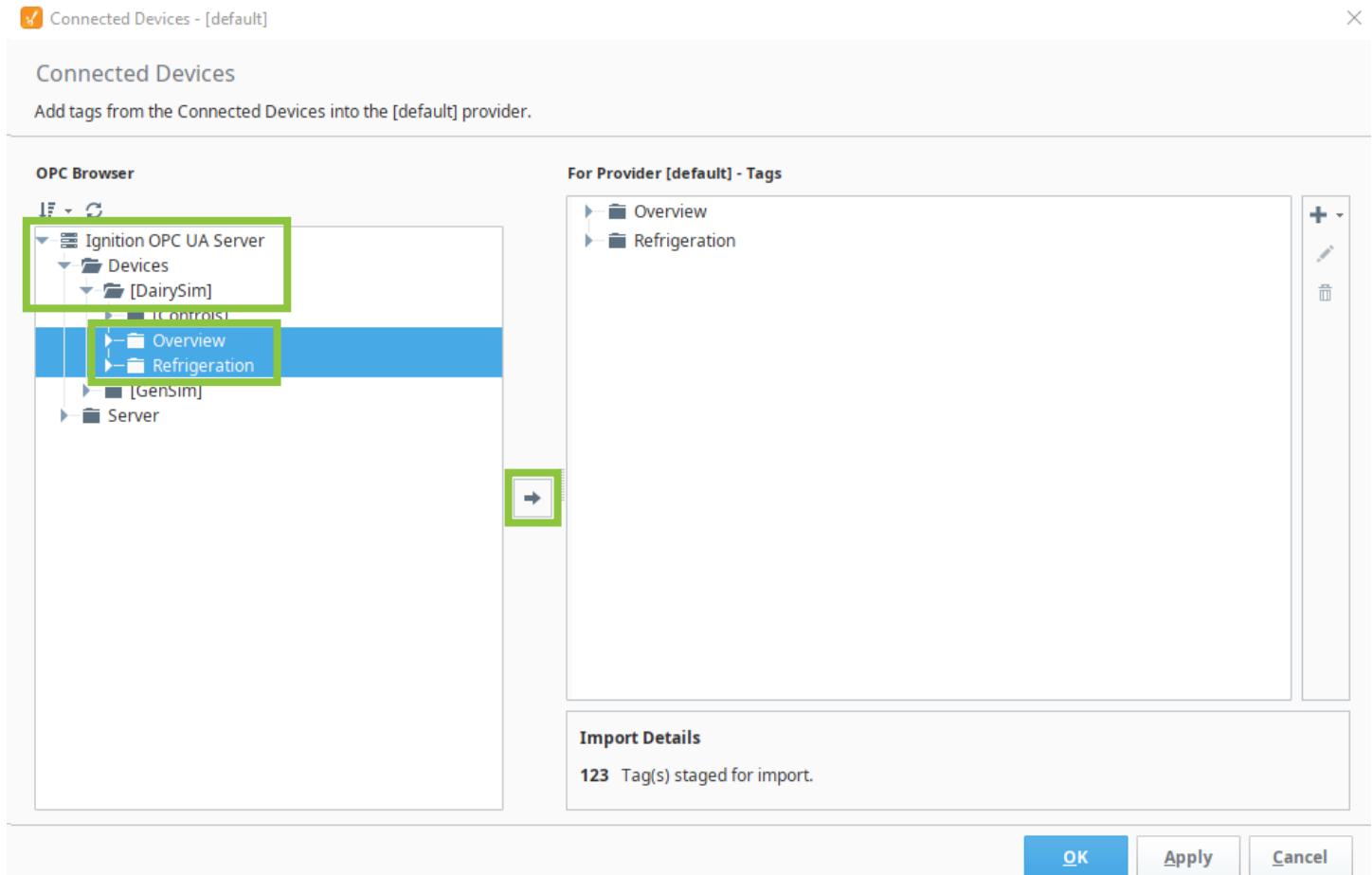
Now that you are in the Designer and have created some navigation windows, a good next step is to add some Tags. You'll use these Tags for real time status and control and to store history with the Tag Historian. Tags are all configured in the Tag Browser panel.

1. Click  in the top left corner of the **Tag Browser**.
2. Click **Browse Devices**.



The Connected Devices window opens.

3. Expand **Ignition OPC UA Server**.
4. Expand **Devices**.
5. Expand **[DairySim]**.
6. Click **Overview**.
7. Hold **Shift**
8. Click **Refrigeration**.
9. Click **Add**.



The Tag folders are added to the **For Provider [default] - Tags** section of the window. The **Import Details** section of the window updates to show that **123** Tags are staged for import.

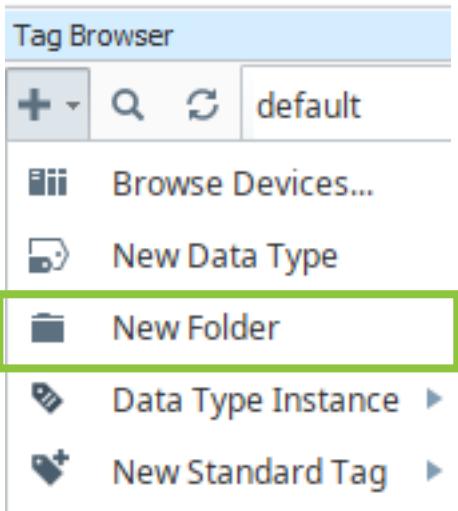
10. Click **OK**.

The Tags are imported and display in the Tag Browser.

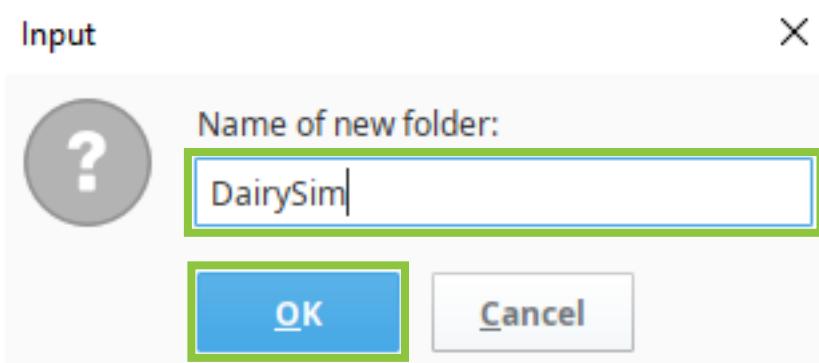
Organize Tags

The Tags are now imported, but each of the two folders are listed individually, and there is no reference to the Diary Sim device that they are coming from. It might make more sense to organize them so that it's clear they are from a common device.

1. Click **+** in the Tag Browser.
2. Click **New Folder**.

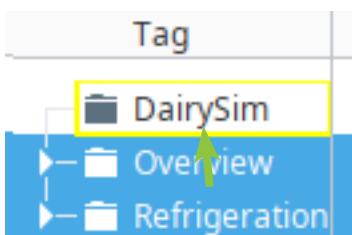


3. Enter **DairySim** in the Name of the new folder field.
4. Click **OK**.



Now that you have the folder created, you can drag the Tag folders to it.

1. Click **Overview**.
2. Press **Shift**.
3. Click **Refrigeration**.
4. Drag both folders to the **DairySim** folder. The folder will highlight in yellow.



5. Click **Yes** to confirm the move.

The Tags are now better organized.

Import Tags Into Folders

In the last section, we imported Tags, then created a folder to organize them. It's possible to save a step and create the folder first, then import the Tags directly into the folder.

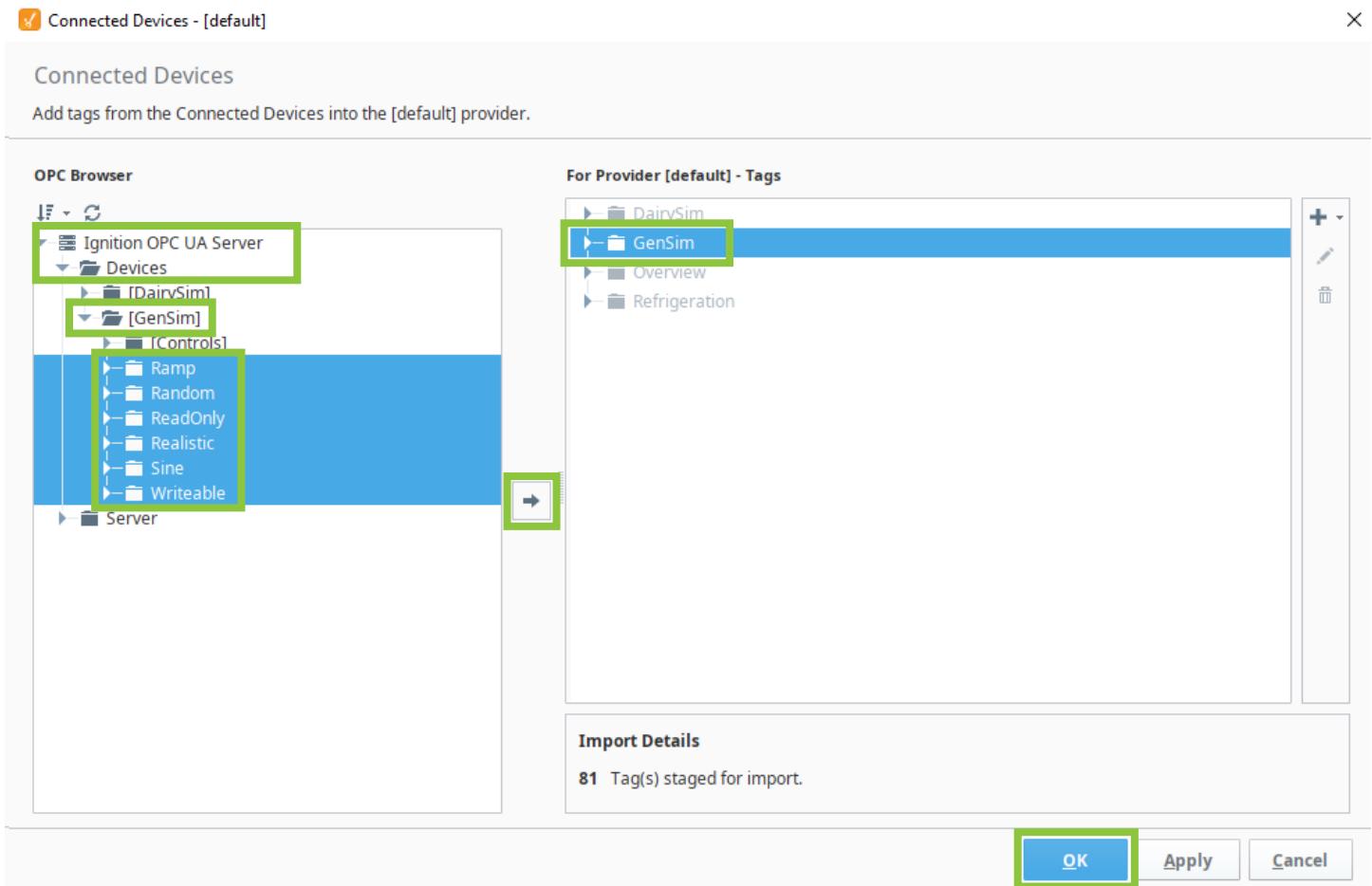
1. Click  in the Tag Browser.
2. Click **New Folder**.
3. Enter **GenSim**.
4. Click **OK**.
5. Click . It is not necessary to have the new folder selected.
6. Click **Browse Devices**.

The Connected Devices window opens.

7. Expand **Ignition OPC UA Server**.
8. Expand **Devices**.
9. Expand **[GenSim]**.
10. Click **Ramp**.
11. Press and hold **Shift**.
12. Click **Writeable**.

Confirm that all six folders—Ramp, Random, ReadOnly, Realistic, Sine and Writeable—are selected.

13. Click **GenSim** in the **For Provider [default] - Tags** section on the right.
14. Click **Add**.



The Tag folders are added to the For Provider [default] - Tags section of the window and automatically placed inside the folder. The Import Details section of the window updates to show that 81 Tags are staged for import.

15. Click **OK**.

The Tags are imported into the folder and display in the Tag Browser.

Note: It may be necessary at times to refresh the Tag Browser if it does not display the most current changes. You can do this by clicking the Refresh button at the top of the Tag Browser.

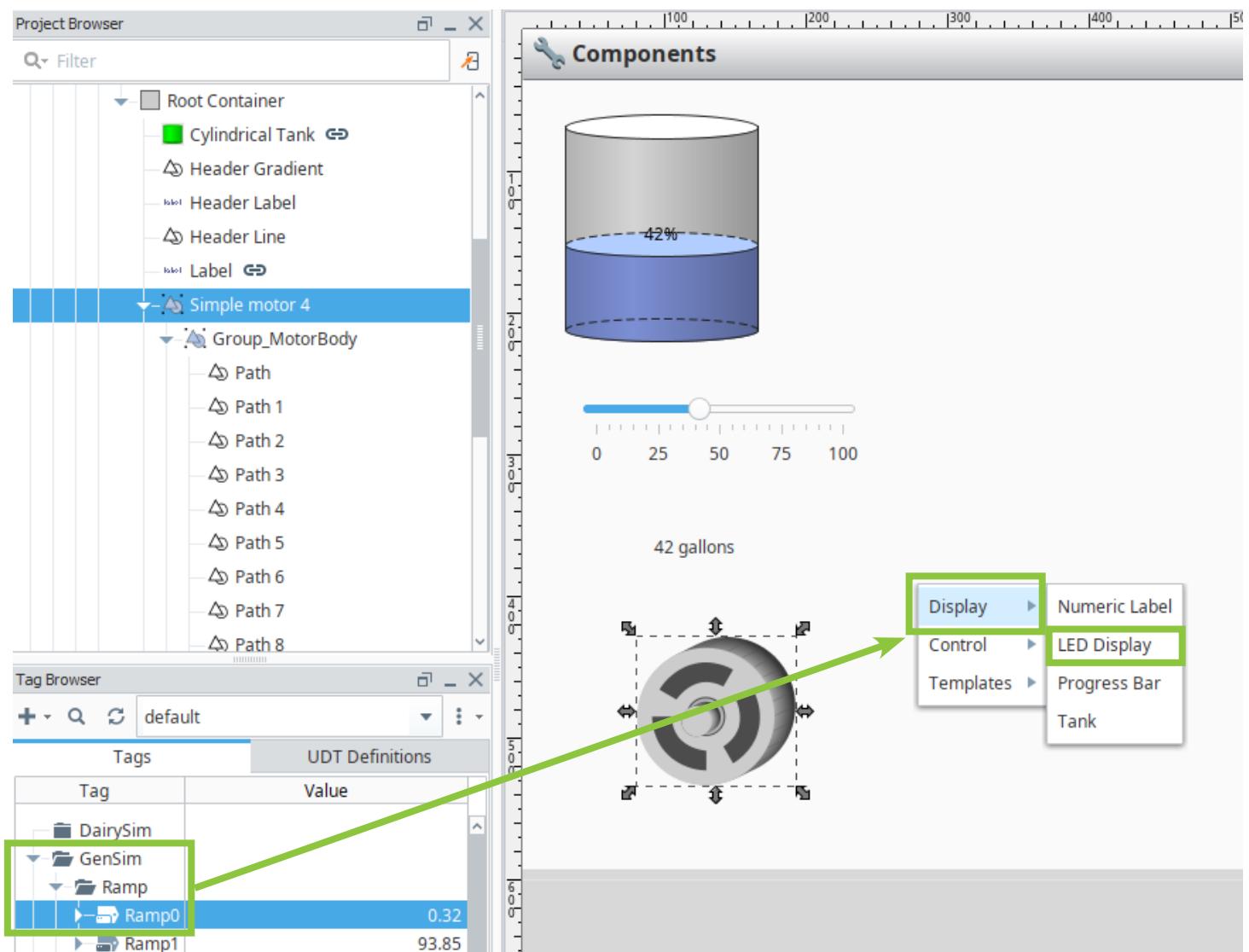
Bind Tags to Components

One of the primary tasks in Ignition is using components to display live Tag data. To do this, you can bind a Tag to a component's property.

Drag a Tag to a Window or View

The simplest way to create a component that visualizes a Tag's value is to directly drag the Tag from the Tag Browser to an open Vision Window or Perspective View.

1. If necessary, open the **Components** Window in Vision.
2. Expand **GenSim** in the Tag Browser.
3. Expand **Ramp**.
4. Drag **Ramp0** to the window.
5. When you release your mouse, you will see a variety of choices for the component to use.
6. Select **Display**.
7. Click **LED Display**.



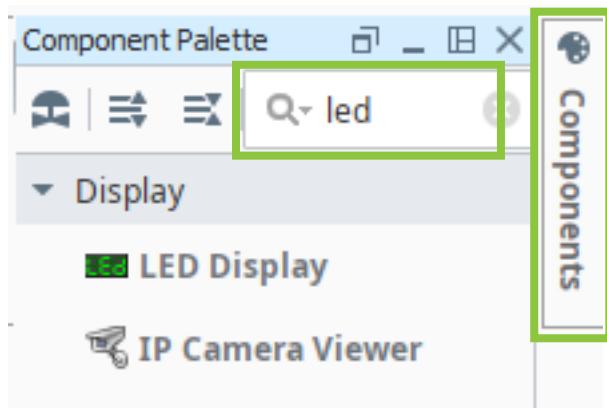
The LED Display component is a simple yet convenient way to display Tag values that are constantly updating.

When dragging a Tag to a window, three component properties are automatically bound to the Tag: **Mouseover Text**, **Number Format Pattern**, and **Value**. You can see this by examining the Vision Property Editor and noting the properties that are bolded and display the Binding icon.

Drag a Tag to an Existing Component

A second method of binding a Tag to a component is to drag the Tag onto a component that already exists in the window or view.

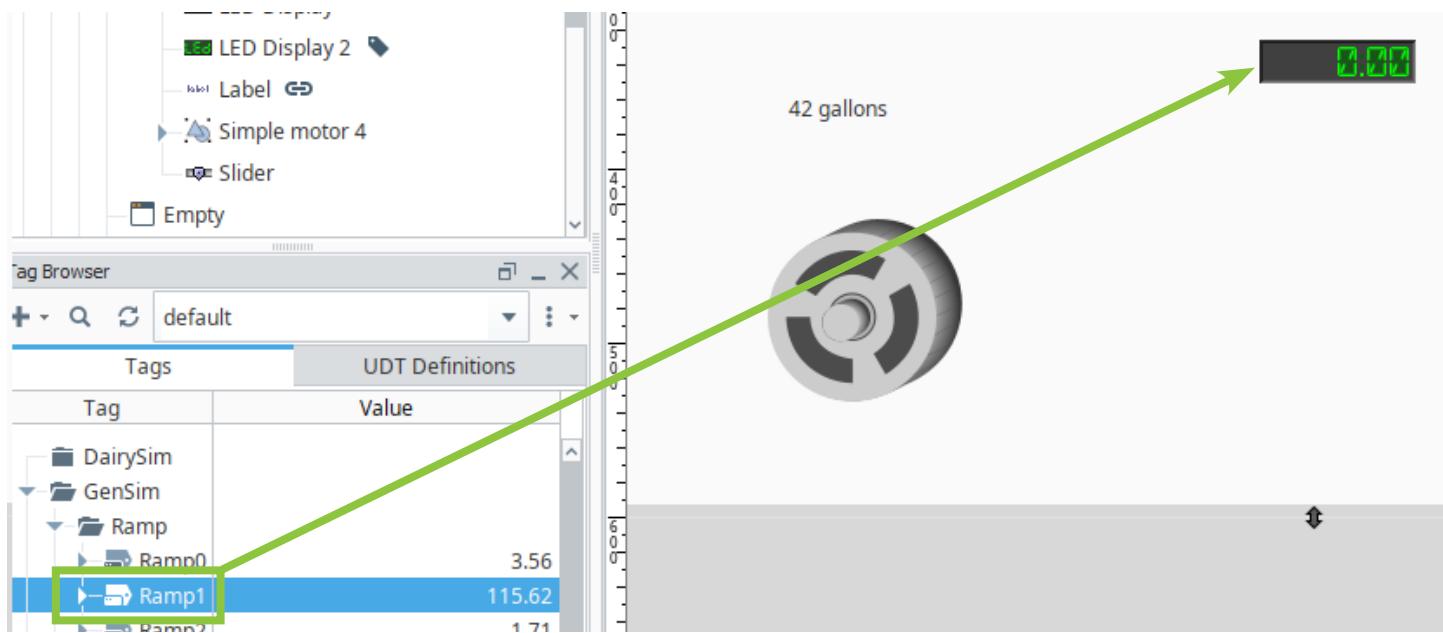
1. Expand the **Components Palette**.
2. Enter **LED** in the filter box at the top of the palette.



3. Drag an **LED Display** component onto the window.



4. Drag **Ramp1** from the Tag Browser onto the new LED component. The component will highlight in yellow.

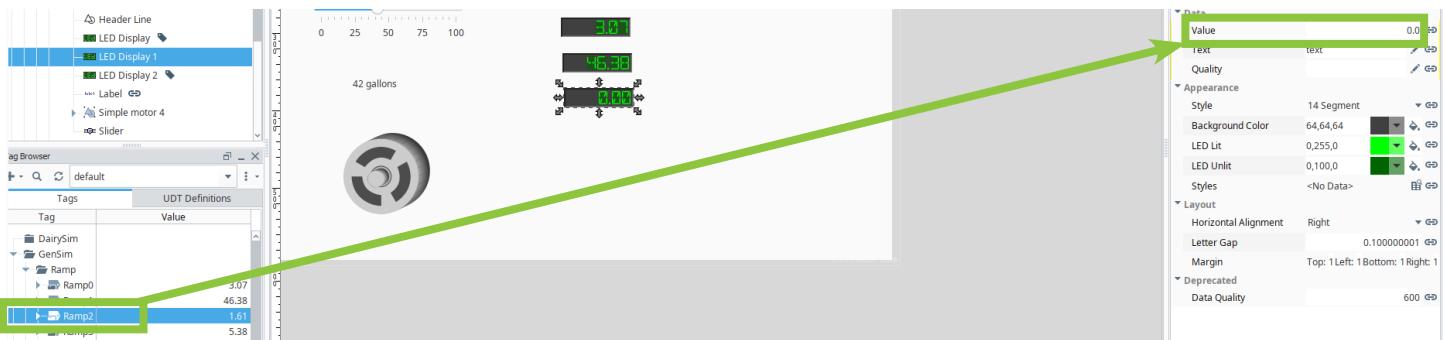


Note that the same three properties—**Mouseover Text**, **Number Format Pattern**, and **Value**—are bound to the component.

Drag a Tag to a Property

A third method for binding a Tag is to drag it directly to a property on the Property Editor of an existing component.

1. Expand the **Components Palette**.
2. Drag an **LED display** component onto the window.
3. Drag **Ramp2** from the Tag Browser onto **Value** property of the new LED component on the Property Editor. The property will highlight in yellow.



Note that only the Value property is bound.

Manually Bind a Property

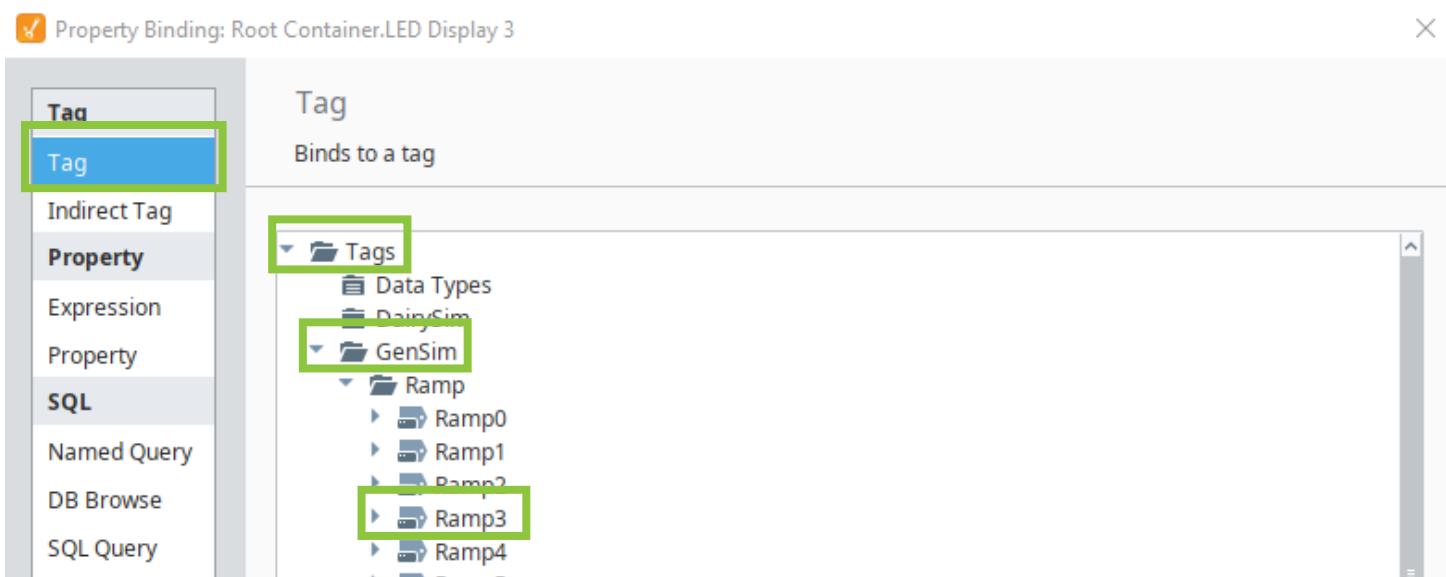
The fourth method for binding a component to a Tag is to manually create the binding.

1. Expand the **Components Palette**.
2. Drag an **LED display** component onto the window.
3. Click  on the **Value** property.



The Property Binding dialog box opens.

4. Click **Tag**.
5. Expand **Tags**.
6. Expand **GenSim**.
7. Expand **Ramp**.
8. Click **Ramp3**.
9. Click **OK**.



Best Practice: Which Method Should You Use?

While all four methods outlined above achieve the same visual result, there is an important difference between the first two and the last two. When dragging a Tag from the Tag Browser, whether directly to the Window or on to a component, you will always end up with three bindings in Vision. But when you drag to a particular property, or manually create the binding, you only get one bound property.

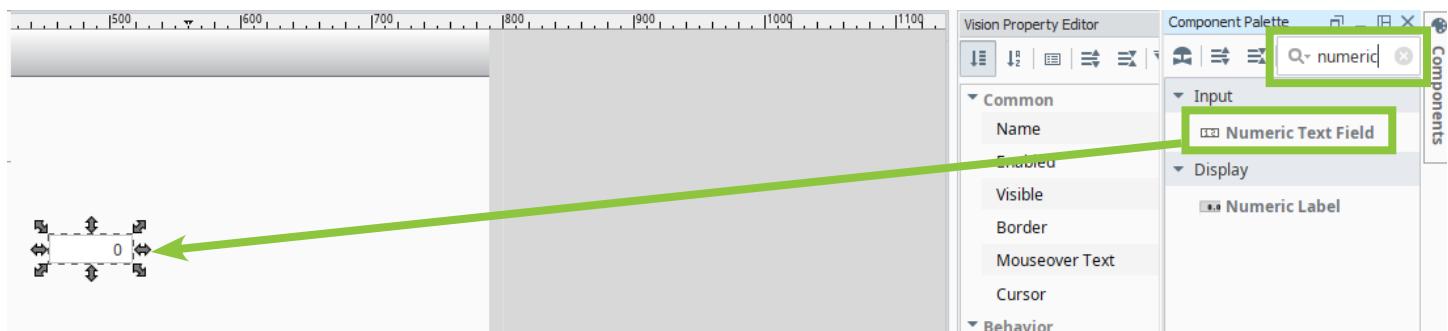
In some cases, this is not going to matter too much, but you need to be aware that those additional bindings might create issues down the road. An example of this would be if you duplicated a component that had the three bindings, but then only changed the Value binding so that the component displayed a different Tag's value. Now, you would have a component that is bound to two different Tags, which might result in unexpected behavior.

For that reason, we recommend primarily using the third or fourth methods over the first two.

Bidirectional Bindings

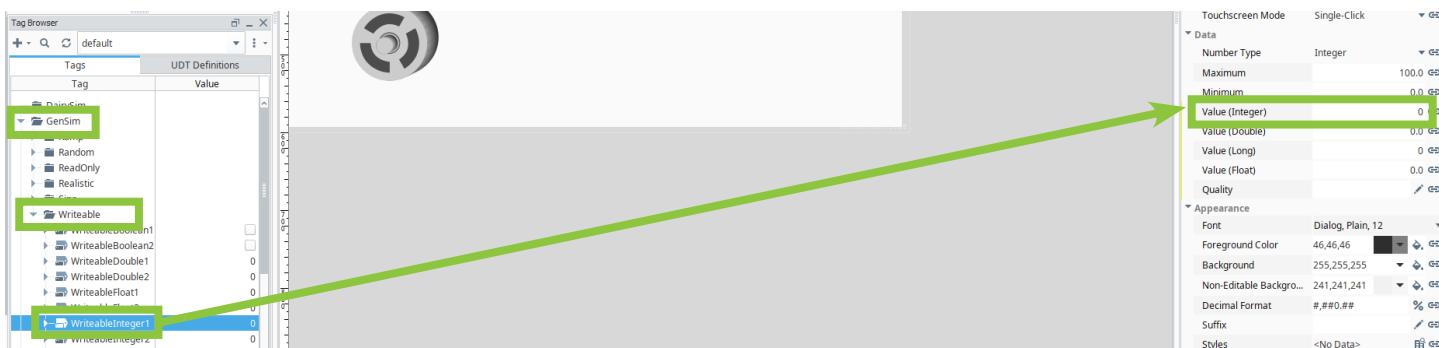
All of the bindings above are read-only, because the LED Display component is itself read-only. But there are many components that are capable of both reading and writing to Tags.

1. Expand the **Components Palette**.
2. Click  to clear the filter.
3. Enter **Numeric**.
4. Drag a **Numeric Text Field** component onto the window.



5. Expand **GenSim** in the Tag Browser.

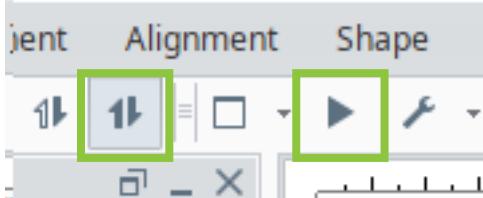
6. Expand **Writable**.
7. Drag **WriteableInteger1** to the Value (Integer) property of the Numeric Text Field.



Dragging a Tag to a writable component automatically creates a bidirectional binding, allowing the component to both read and write the Tag's value.

We can test this behavior using Preview mode.

8. Click ►.
9. Click **Read/Write** on the main toolbar.



10. Enter a **number** into the field.

11. Press **Enter**.

If you examine the Tag's value in the Tag Browser, you will see it now has the new value.

Memory Tags

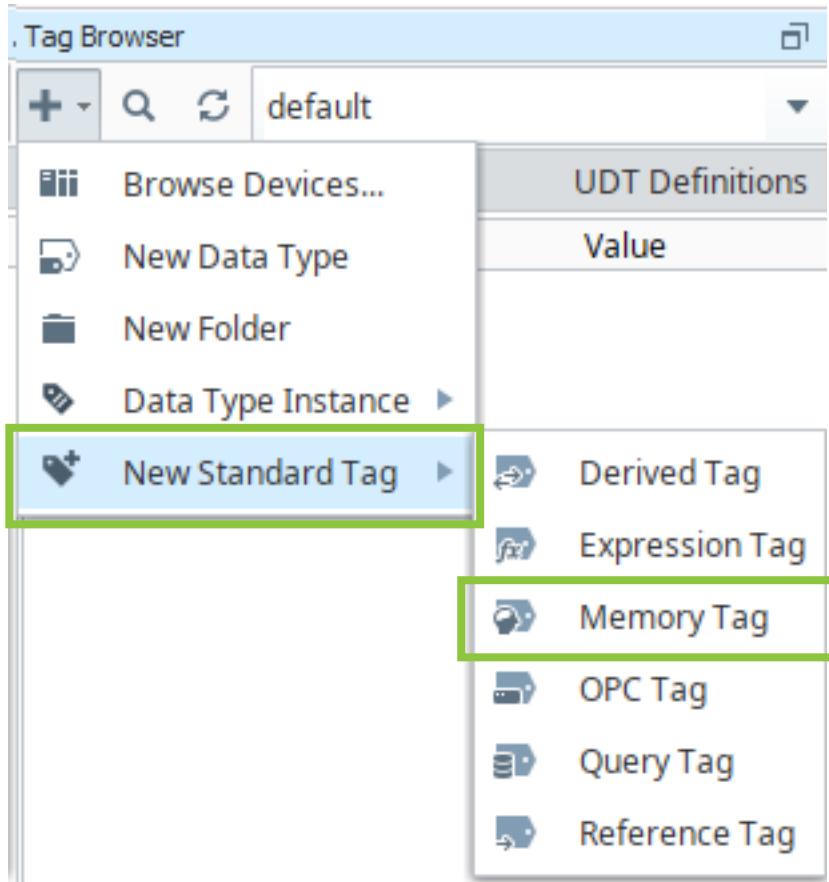
All of the Tags we have explored so far have been OPC Tags—Tags that represent values from actual (or, in our case, simulated) PLCs.

However, there are other types of Tags available to us in Ignition. A common and very useful example of this is a Memory Tag. Memory Tags are similar to global variables in programming, as they allow us to store a value that can be used later, in any part of our project. In fact, because Memory Tags, like all other Tags, are stored on the Gateway, they are available to every project on that Gateway.

Create a Memory Tag

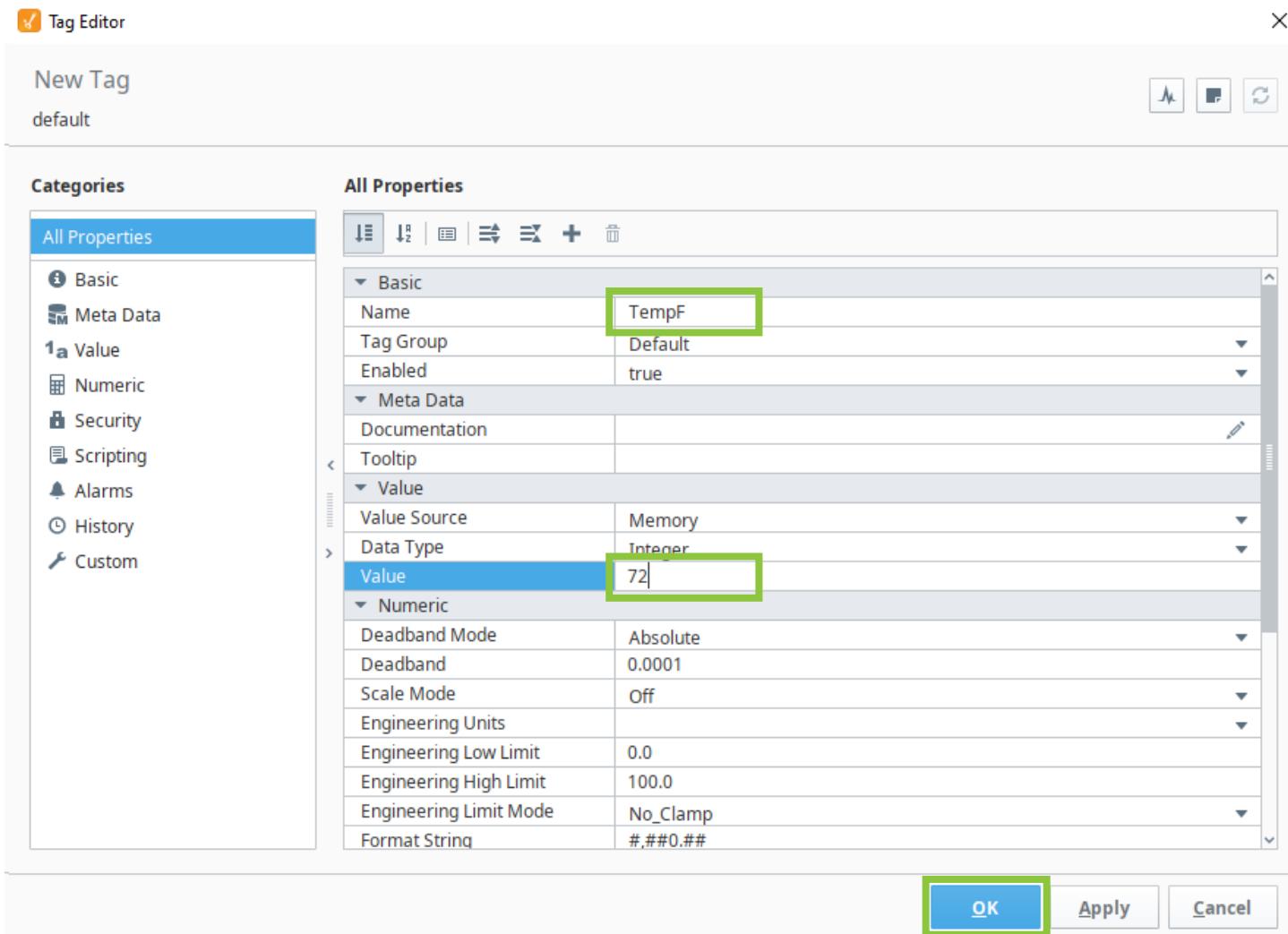
Memory Tags are created in the Tag Browser.

1. Collapse any open folders in the Tag Browser.
2. Click  in the Tag Browser.
3. Click **New Standard Tag**.
4. Click **Memory Tag**.



The Tag Editor opens.

5. Enter **TempF** in the Name field.
6. Enter **72** in the Value field.
7. Click **OK**.



The Memory Tag is created with a default value.

Derived Tags

One use of a Memory Tag is to use it to drive the value of a Derived Tag.

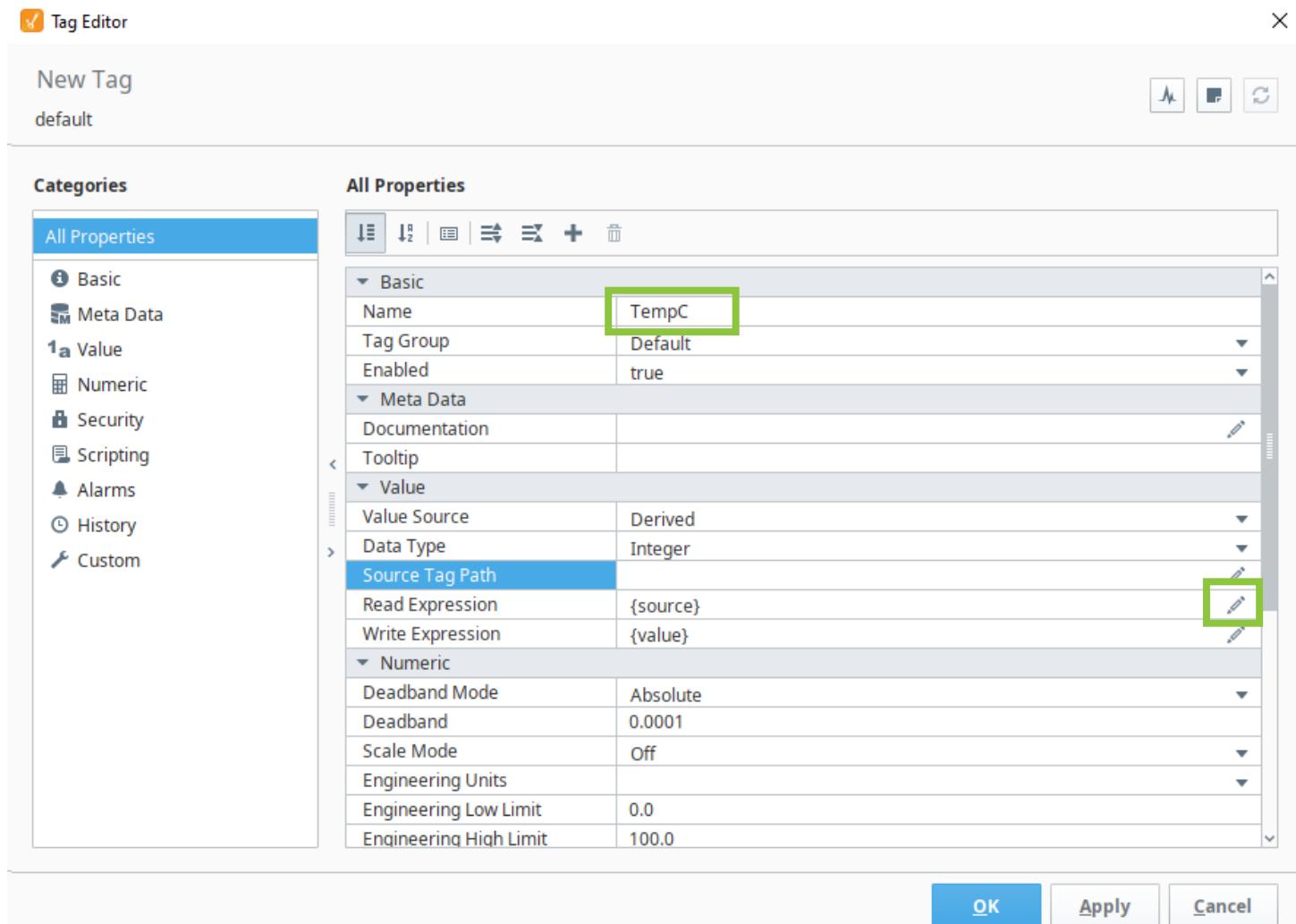
Derived Tags have a source—some other Tag—as well as read and write expressions. This way, a Derived Tag can both get a value based on its source, but also write back to the source.

1. Collapse any open folders in the Tag Browser.
2. Click in the Tag Browser.
3. Click **New Standard Tag**.
4. Click **Derived Tag**.

The Tag Editor opens.

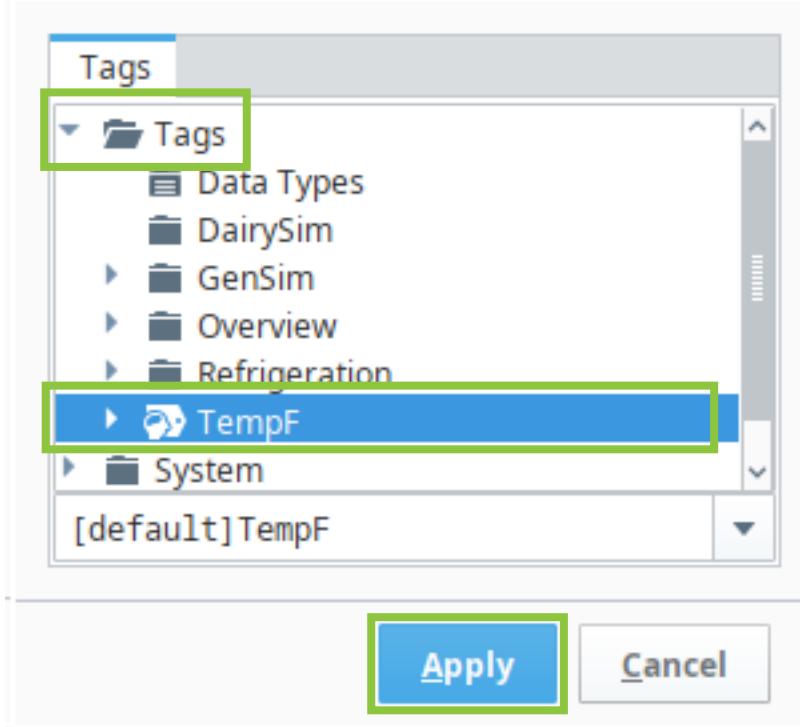
5. Enter **TempC** in the Name field.

6. Click  on the **Read Expression** property.



The Expression dialog box opens.

7. Expand **Tags**.
8. Click **TempF**.
9. Click **Apply**.



10. Click on the **Read Expression** property.

The Expression Editor opens. We need to write an expression to tell this Tag how to calculate its value from its Source Tag.

11. Enter the following expression to convert a temperature from Fahrenheit to Celsius:

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

12. Click **Apply**.

13. Click on the **Write Expression** property.

When this Tag changes, it needs to write a value back to the Source Tag.

14. Enter the following expression to convert from Celsius to Fahrenheit:

```
{value}*9/5+32
```

15. Click **Apply**.

16. Click **OK**.

You will now see the two new Tags, and will notice that TempC—the Derived Tag—has a value of 22, the Celsius equivalent of 72 degrees Fahrenheit.

You can test the functionality of both Tags by changing either value.

Additional Tag Types

Several other types of Tags exist on the Gateway.

Expression Tag A Tag 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.

Query Tag Executes a SQL Query, whose result provides the value for the Tag. Like SQL binding in Vision, SQL Query Tag can reference other Tags to build dynamic queries.

Reference Tag A simple type that has the same value as another Tag. This way you can have multiple copies from the same source, and set different values for alarms, history, scaling, etc.

Complex Tag (UDT) Created from a standard Tag type, but offers 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 Tag is defined and can then be created as if it were a single Tag.

System and Vision Client Tags

Ignition comes with predefined system Tags to provide more information about the Client and the Gateway System.

You can switch between the Tags derived from devices and the Client or System Tags by selecting the appropriate Tag source in the list at the top of the Tag Browser.

System Tags

System Tags provide status about the system, such as memory usage, performance metrics, and so on. They exist for the Client and the Gateway. You can modify the Gateway system Tags to perform alarming, history, and scaling, but you cannot modify the Client Tags.

Vision Client Tags

Client Tags, as the name implies, are only available for use in Vision Clients. 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. However, they do not have a Tag Group. When set to run as an expression or query, a poll rate is specified dictating how often the value is calculated.

A great way to use Client Tags is to have a variable that is used across multiple pages when you can't use a normal Memory Tag because each running Client would want their own value. For example, you might have several windows reused for all areas around your facility. If you are passing the area name into each screen, then you want to use a Tag to remember without having to pass variables around while opening windows. If two or more people would want to see different areas at the same time, a Memory Tag won't work but a Client Tag will.

Tag Groups

Tags are executed by Tag Groups inside of a Tag Provider.

Tag Groups dictate the rate of execution of Tags, and therefore play crucial role in the design of large and high-performance systems. They offer several key execution modes to help create efficient projects.

Direct Tag Group

The Direct Mode executes at a fixed rate, which is defined by the Rate setting. Every Tag that uses the Direct mode will poll the PLC at the specified rate setting.

Ignition comes with two default Direct Tag Groups: Default and Default Historical. The Default group polls at a 1 second rate, while the Default Historical group polls at a 10 second rate. However, you can modify either of these if you need to.

Leased Tag Group

The leased Tag Group executes at two rates. A Leased/Driven Rate executes when a component on any open window, in either Designer or the Client, is bound to the Tag. If no components are on open windows bound to the Tag, it will poll at the Rate setting. Usually, the Rate will be slower than the Leased/Driven Rate.

Driven Tag Group

The rate of the Driven Tag Group is based on a simple comparison between a Tag value and a number. If the condition is true, the Tag Group executes at its Leased/Driven rate. If false, it runs at the Rate.

Create New Tag Groups

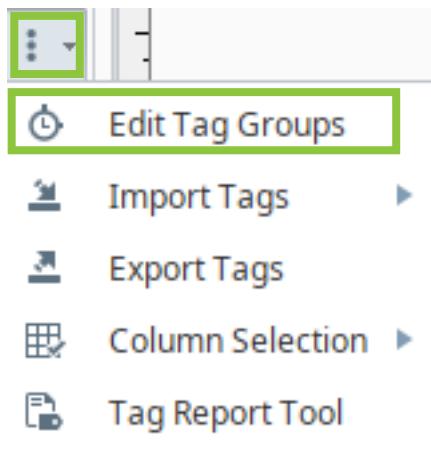
For the purposes of seeing how Tag groups work, we will create two new Tag groups so that we can explore how Leased and Driven groups work.

Create New Tag Groups

We want to create a Leased Tag Group that will update Tags once per second when a component on an open window is bound to them, and once every ten seconds otherwise.

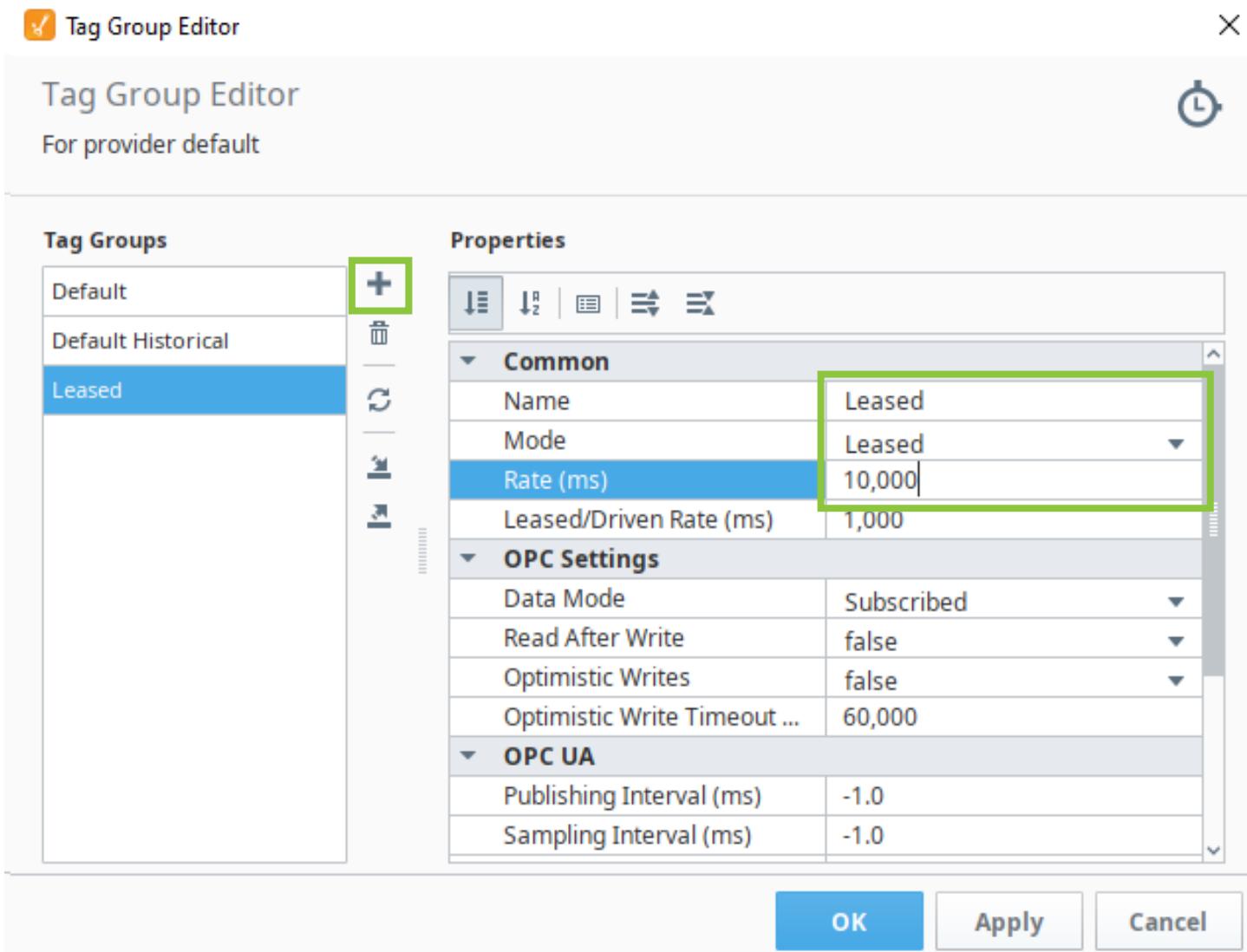
Then, we will create a Driven Tag Group that will update Tags once per second if the value of our TempF Memory Tag is above 32, and not update their values at all if it is below 32.

1. Click  in the top right corner of the Tag Browser.
2. Click **Edit Tag Groups**.

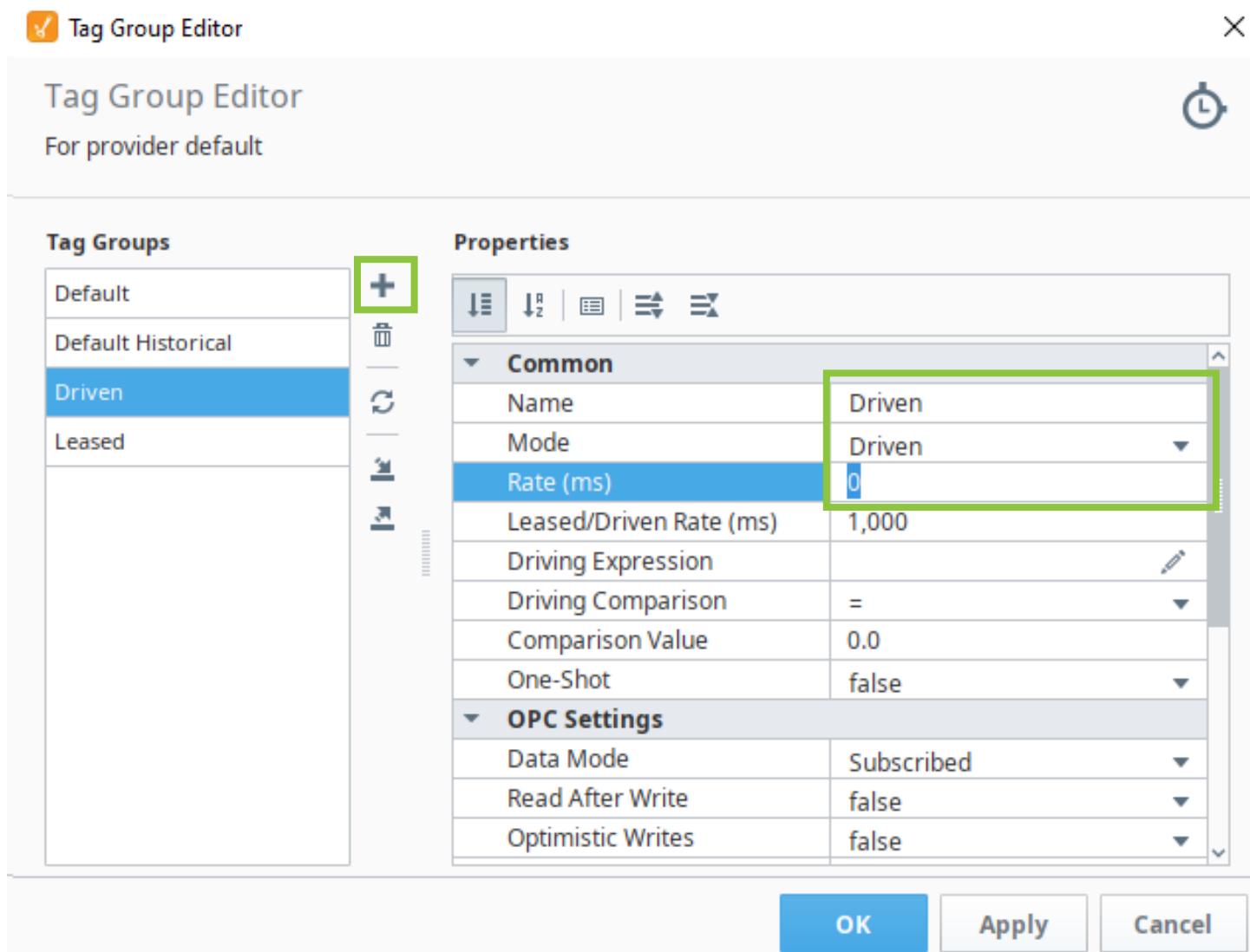


The Tag Group Editor opens.

3. Click  to add a new Tag Group.
4. Enter **Leased** in the Name field.
5. Select **Leased** from the Mode list.
6. Enter **10000** in the Rate (ms) field.



7. Click to add a new Tag Group.
8. Enter **Driven** in the Name field.
9. Select **Driven** from the Mode list.
10. Type **0** in the Rate (ms) property.

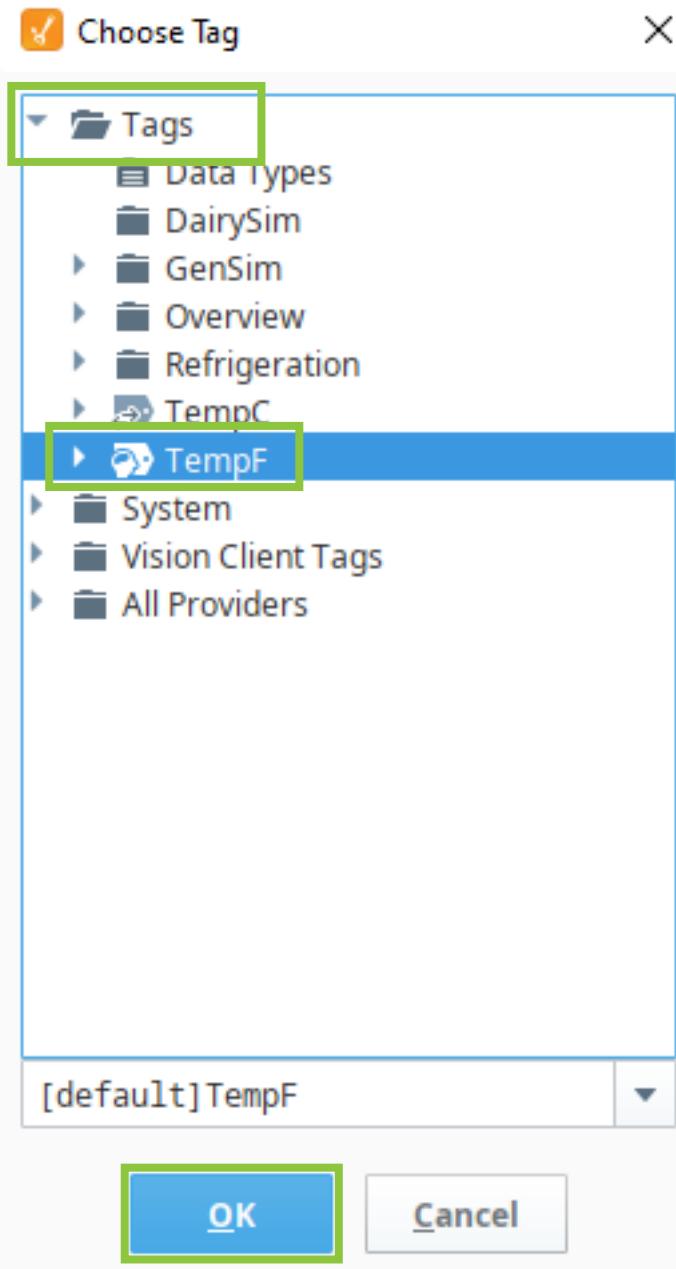


Setting a Rate to zero means that Tags will never poll for a new value if the expression is not met.

11. Click to edit the **Driving Expression** property.

We want an expression that will first read our TempF memory Tag. We can use the dialog box to find and correctly insert the path to that Tag.

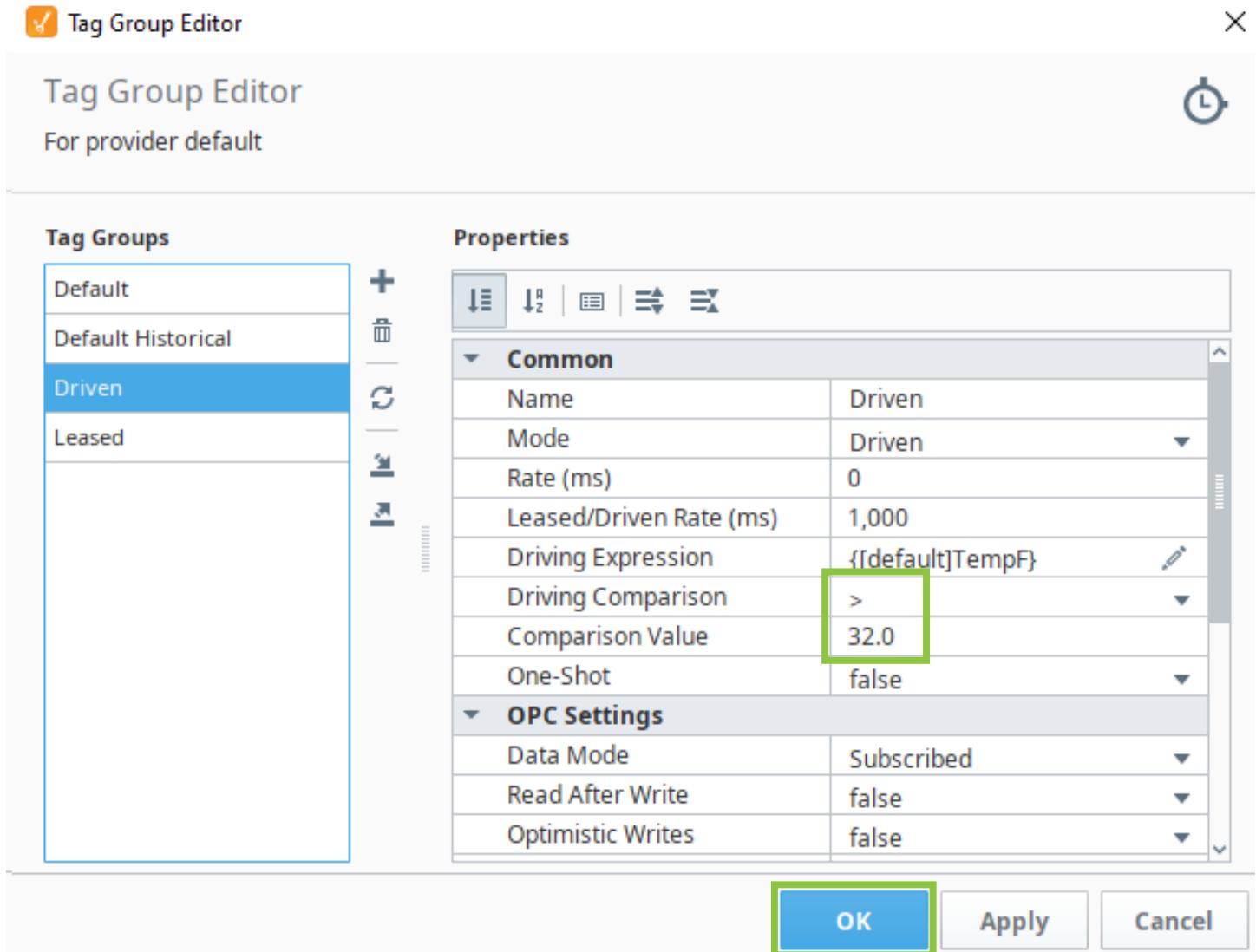
12. Click .
13. Expand **Tags**.
14. Click **TempF**.
15. Click **OK**.



16. Click **Apply**.

A reference to the Tag, **{[default]TempF}**, is inserted into the Driving Expression property.

17. Select **>** from the Driving Comparison list.
18. Type **32** in the Comparison Value field.
19. Click **OK**.



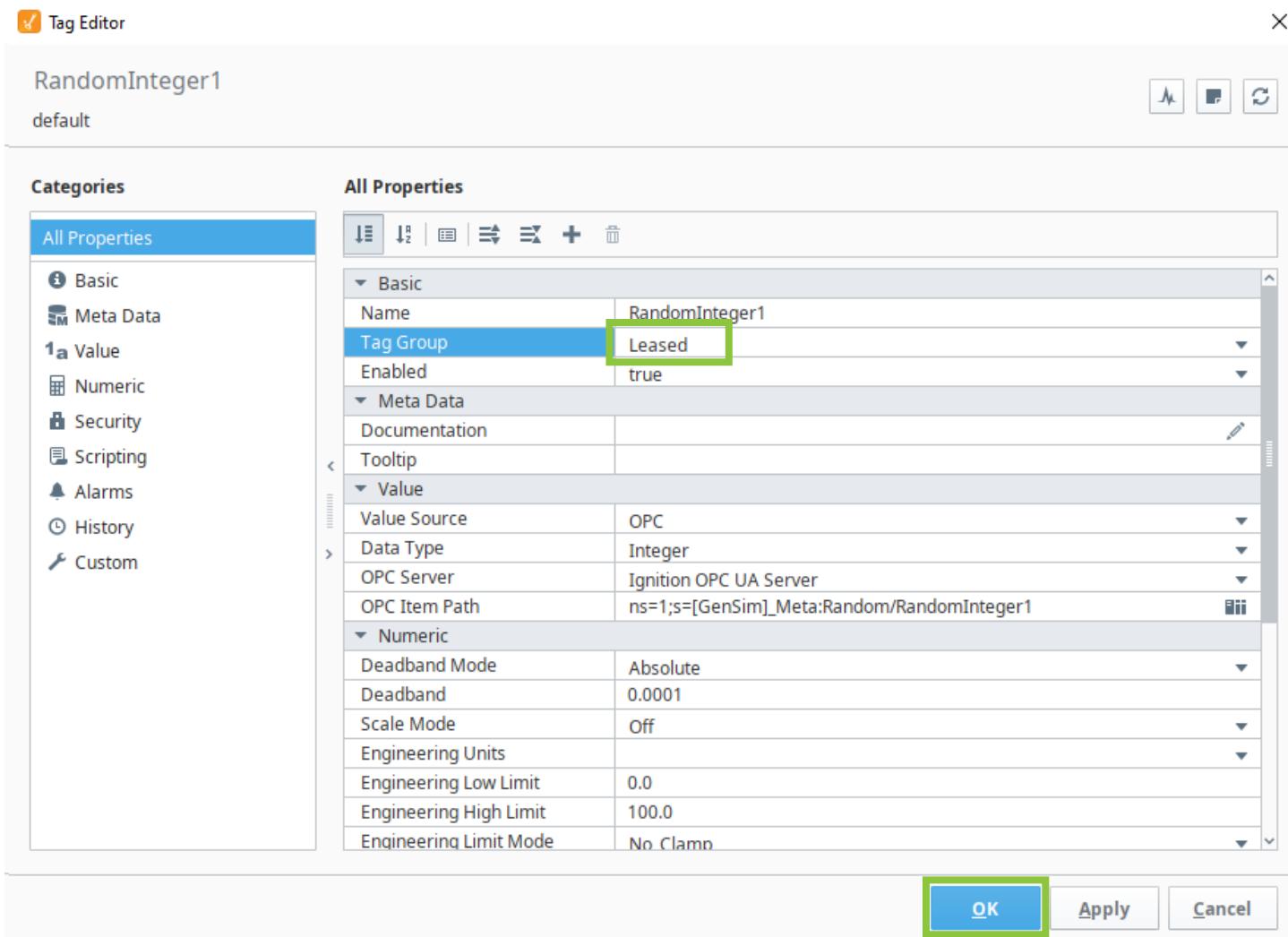
Apply Tag Groups to Tags

Now that we have the Tag Groups, we want to apply them to some Tags.

1. Expand **GenSim** on the Tag Browser.
2. Expand **Random**.
3. Double-click **RandomInteger1**.

The Tag Editor opens.

4. Select **Leased** from the Tag Group dropdown.
5. Click **OK**.



Because we have set it to the Leased Tag Group, but have no components on any windows bound to the Tag, it will now only update once every 10 seconds.

6. Drag **RandomInteger1** to the window.
7. Select **Display**.
8. Click **LED Display**.

Now that there is an open window with a component displaying the Tag's value, it will begin updating every second.

9. Double-click **RandomInteger2**.
10. Select **Driven** from the Tag Group dropdown.
11. Click **OK**.
12. Double-click the value of **TempF**.

Ch 8. Tags

13. Type **50**.
14. Press **Enter**.
15. The value of **RandomInteger2** will update once every second.
16. Double-click the value of **TempF**.
17. Type **20**.
18. Press **Enter**.

The value of **RandomInteger2** will stop updating.

Complex Tags (UDTs)

Complex Tags, also referred to as Tag UDTs, let you leverage object-oriented data design principles in Ignition. Using complex Tags, you can dramatically reduce the amount of work necessary to create robust-systems by essentially creating parameterized “data templates” that you can use to generate Tag instances.

UDT Features

UDTs allow you to more efficiently manage large groups of Tags. Some key features of UDTs include:

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 Tag, 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 your member Tags. When its 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 flexibility for dealing with irregular data and corner cases.

Terminology

The following terms are frequently used when discussing complex Tags:

UDT (User Defined Type) The definition of the data type: its structure, Tags, attributes, and settings.

Instances Running copies of a data type with specific data for all members. Instances are linked to their parent types and are refreshed when their parent type changes. Except for specifically overridden settings, all settings are inherited from the type definition. This should be a specific object, like Motor 3, or Cooling Tank 5.

Parameters 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 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.

Members Tags inside of a data type or instance. Members can be basic Tag types, or other UDTs.

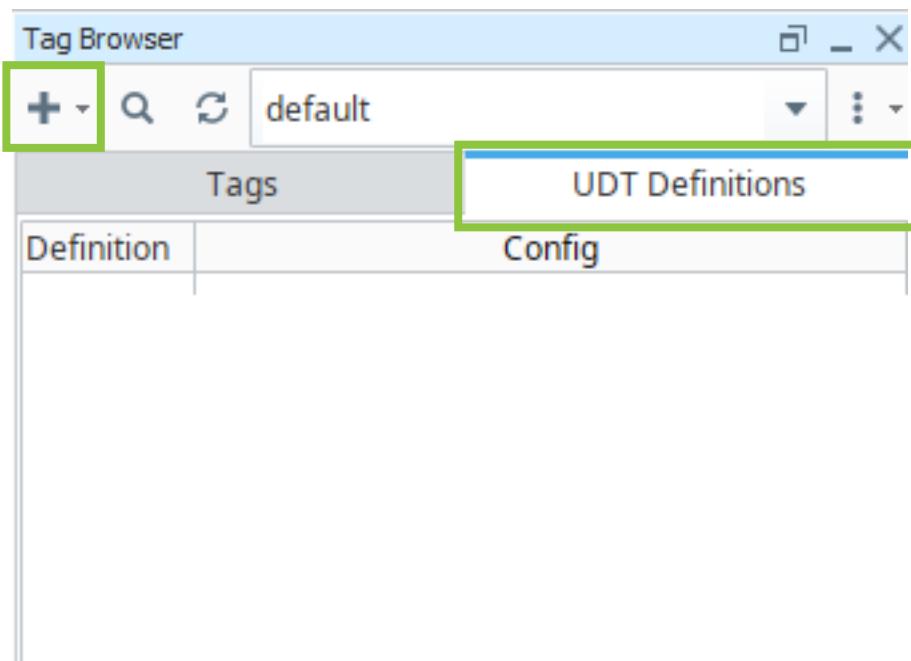
Create a UDT

Our DairySim device contains eight Motors, each with an identical structure. These are a prime candidate for a set of Tags that could be more easily managed as an UDT.

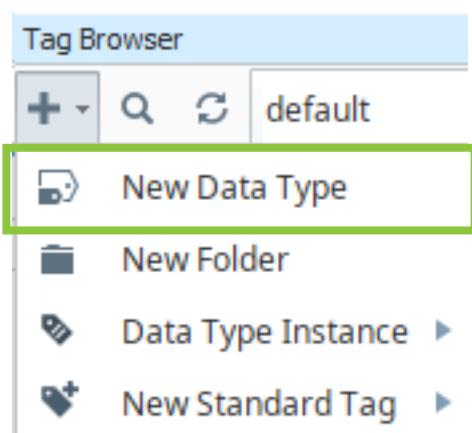
Creating a MotorUDT

The easy way to create a UDT is to model it after an existing sample Tag. So, we will create a UDT for our Motors, and initially base it off of one of the existing Motor Tags.

1. Click **UDT Definitions** in the Tag Browser.
2. Click **+ ▾** in the top left corner of the Tag Browser.

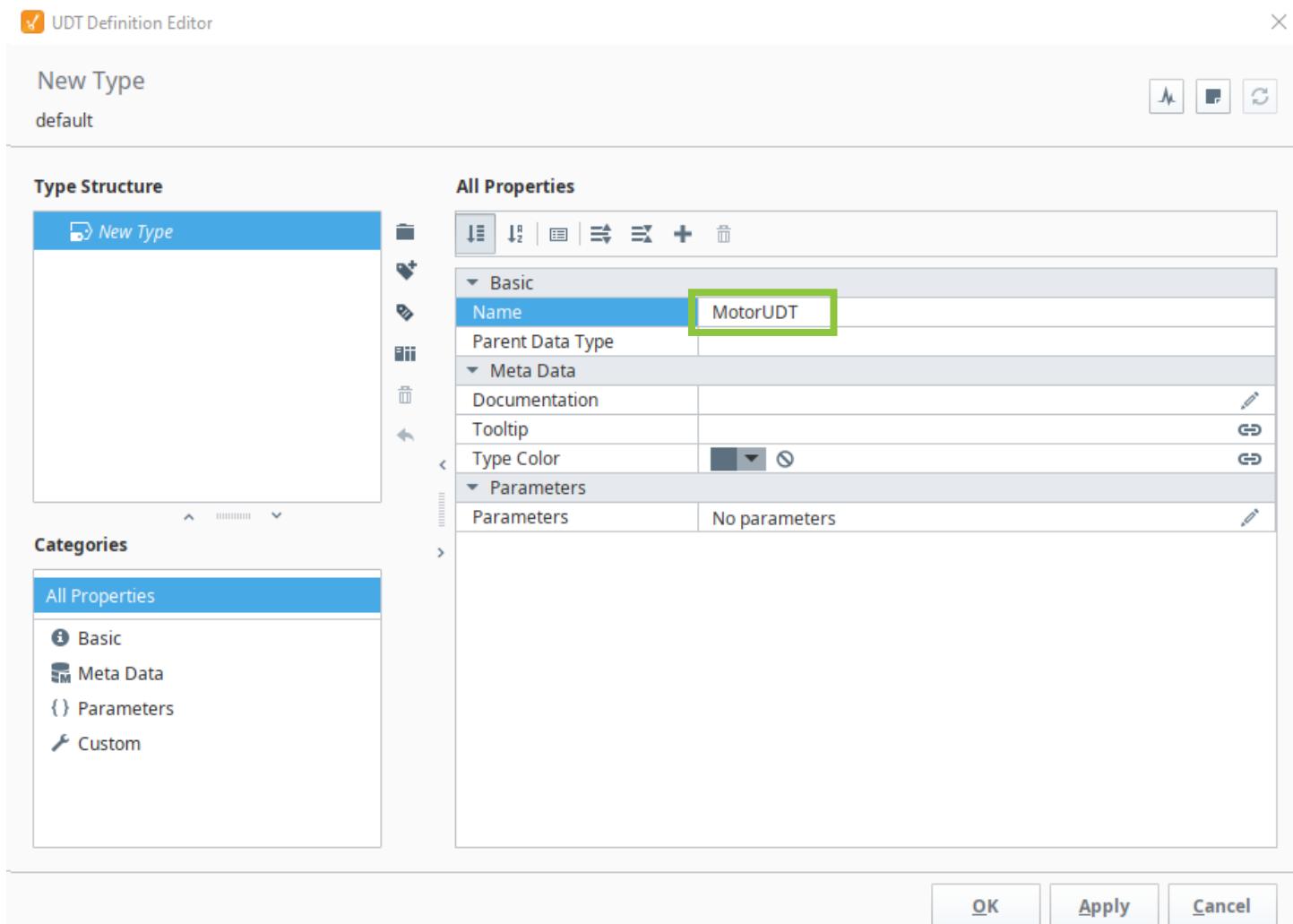


3. Click **New Data Type**.



The UDT Definitions Editor opens.

4. Enter **MotorUDT** in the Name field.

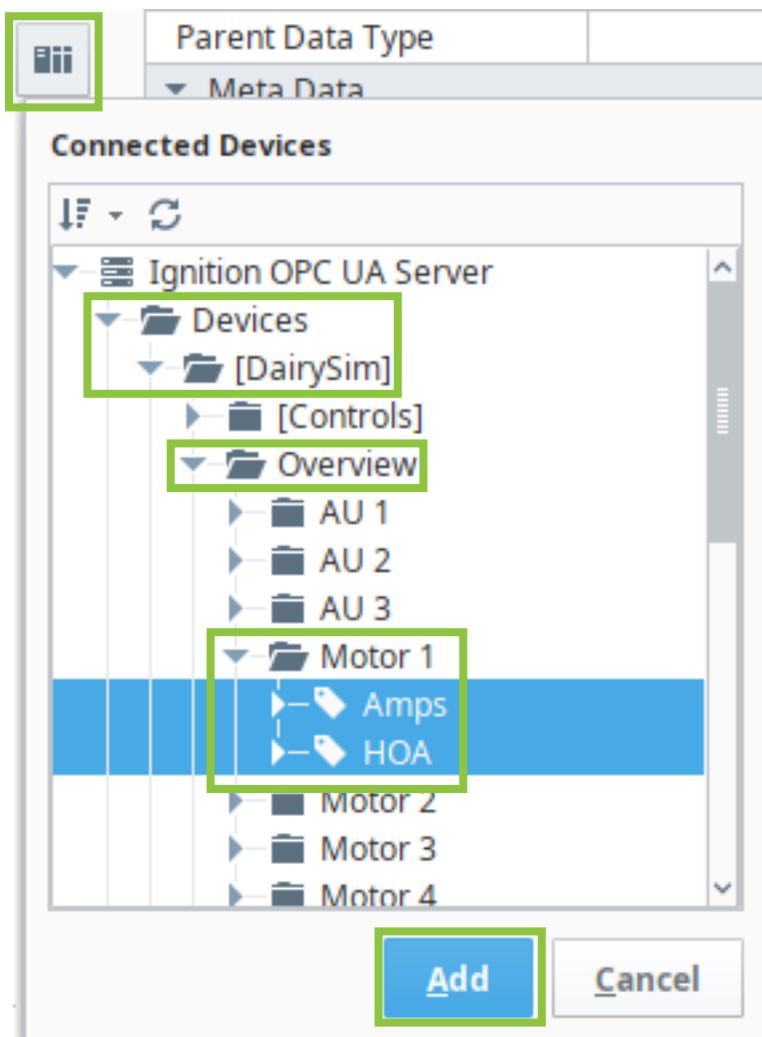


The UDT will need references to the Tags we want to include in it. We can create these by adding sample Tags, then changing them to be general references to the Tag type instead.

5. Click
6. Expand **Ignition OPC UA Server**.
7. Expand **Devices**.
8. Expand **[DairySim]**.
9. Expand **Overview**.
10. Expand **Motor 1**.
11. Click **Amps**.
12. Press **Shift**.

13. Click **HOA**.

14. Click **Add**.



15. Leave the UDT Definition Editor open.

UDT Parameters

All of the Tags we have worked with up until now have had a hard-coded OPC item path. For example, the path for the Amps Tag in Motor 1 looks like this:

```
ns=1;s=[DiarySim]_Meta:Overview/Motor 1/Amps
```

And the path for Motor 2:

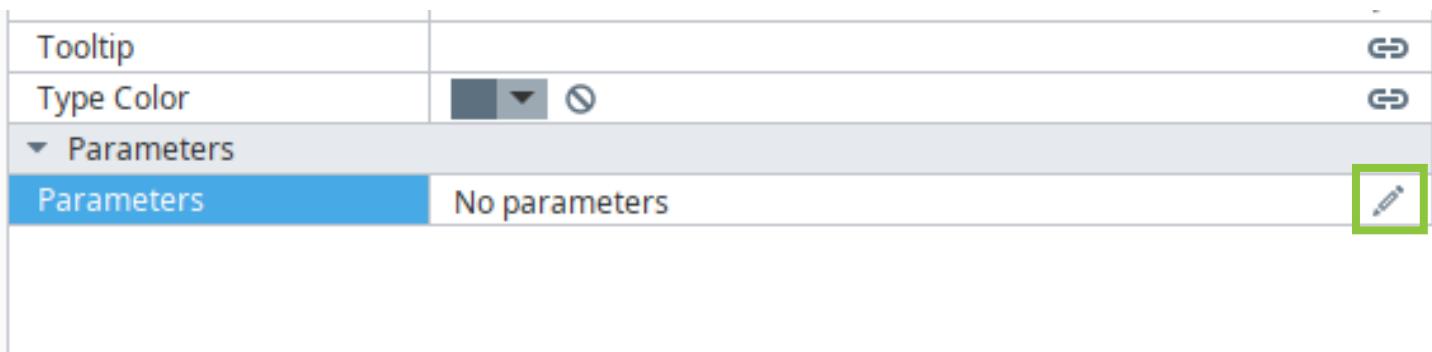
```
ns=1;s=[DiarySim]_Meta:Overview/Motor 2/Amps
```

If we carefully examine those paths, we can see that they are identical, with a single exception: the number following “Motor”.

Our UDT needs to be able to represent the generic idea of a Motor, but the individual instances of Motors we will create from it will need a way to differentiate between those paths. It is able to do this with a parameter—a value we can pass in to each instance as a variable.

Note: In our simplified example for class, we are using a single parameter to represent a single portion of the OPC Item Path. In real-world situations, though, UDTs might require multiple parameters that represent a variety of values within the UDT.

1. Click  on the Parameters property.



2. Click  to add a parameter.
3. Click in the Name field.
4. Enter **MotorNumber**.

Note: The spacing and capitalization in the parameter name is not important, but every time you need to reference this parameter, it will need to be spelled (including capitalization) exactly the same way, so it's important to have a naming convention established to ensure consistency. For class, we will always spell multi-word parameters as one word, capitalizing the first letter of each word.

5. Press **Enter**.

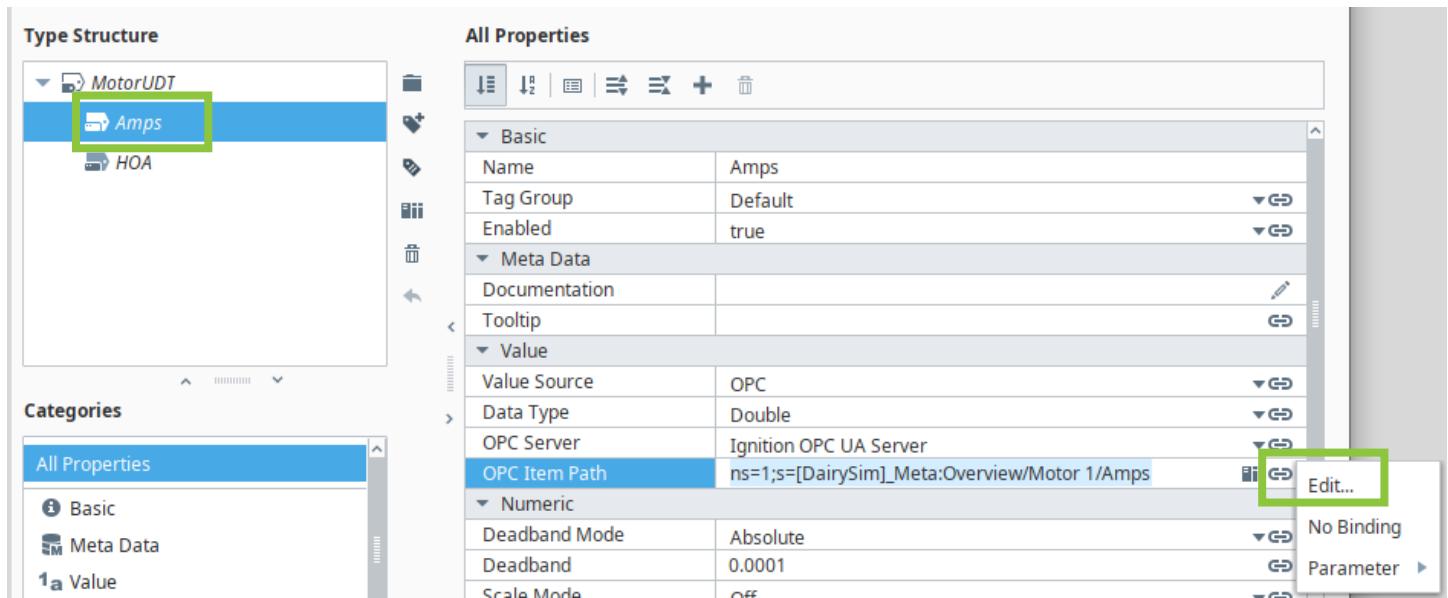
Parameters		
Name	Data Type	Value
MotorNumber	integer	

- Leave the UDT Definition Editor open.

Using the Parameter

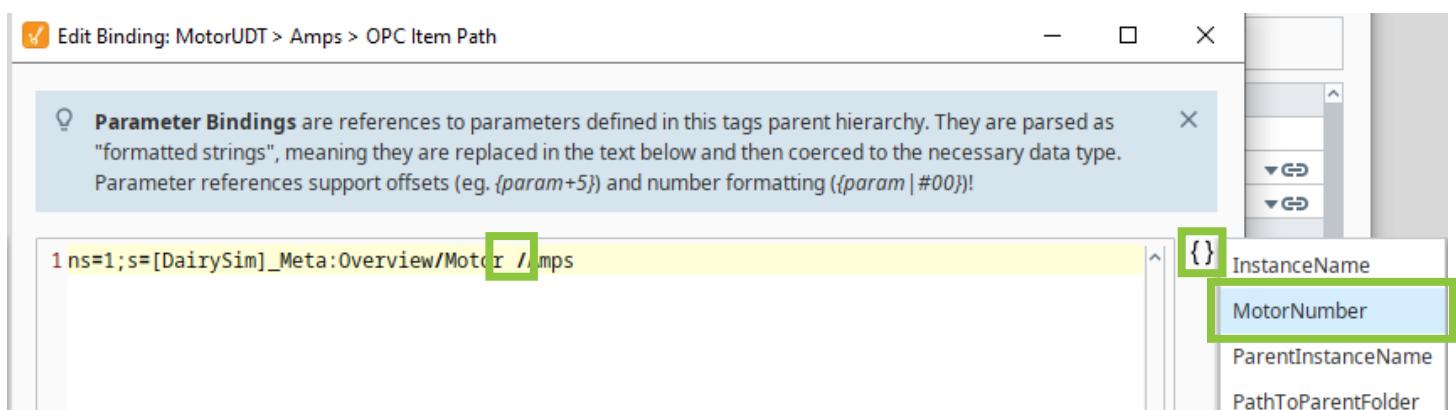
Now that we have created the parameter, we need to edit the OPC Item Path for each Tag in our UDT.

- Click **Amps**.
- Click  on the OPC Item Path property.
- Click **Edit**.



The Edit Binding dialog box opens.

- Delete the **1** in the path.
- Click **{}**
- Click **MotorNumber**.



A reference to the parameter, **{MotorNumber}**, is inserted into the path.

7. Make sure your path now looks like this:

```
ns=1;s=[DiarySim]_Meta:Overview/Motor {MotorNumber}/Amps
```

Note: Be sure there is a space between the word “Motor” and the parameter reference.

8. Click **Apply**.
9. Click **HOA**.
10. Click  on the OPC Item Path property.
11. Click **Edit**.

The Edit Binding dialog box opens.

12. Delete the **1** in the path.
13. Click **{}**
14. Click **MotorNumber**.

A reference to the parameter is inserted into the path.

15. Make sure your path now looks like this:

```
ns=1;s=[DiarySim]_Meta:Overview/Motor {MotorNumber}/HOA
```

16. Click **Apply**.
17. Click **OK**.

Create Instances of the UDT

Now that we have a UDT created, we can make instances of it. Instances are running copies of a data type with concrete data for all members.

We'll start by creating a folder for our instances.

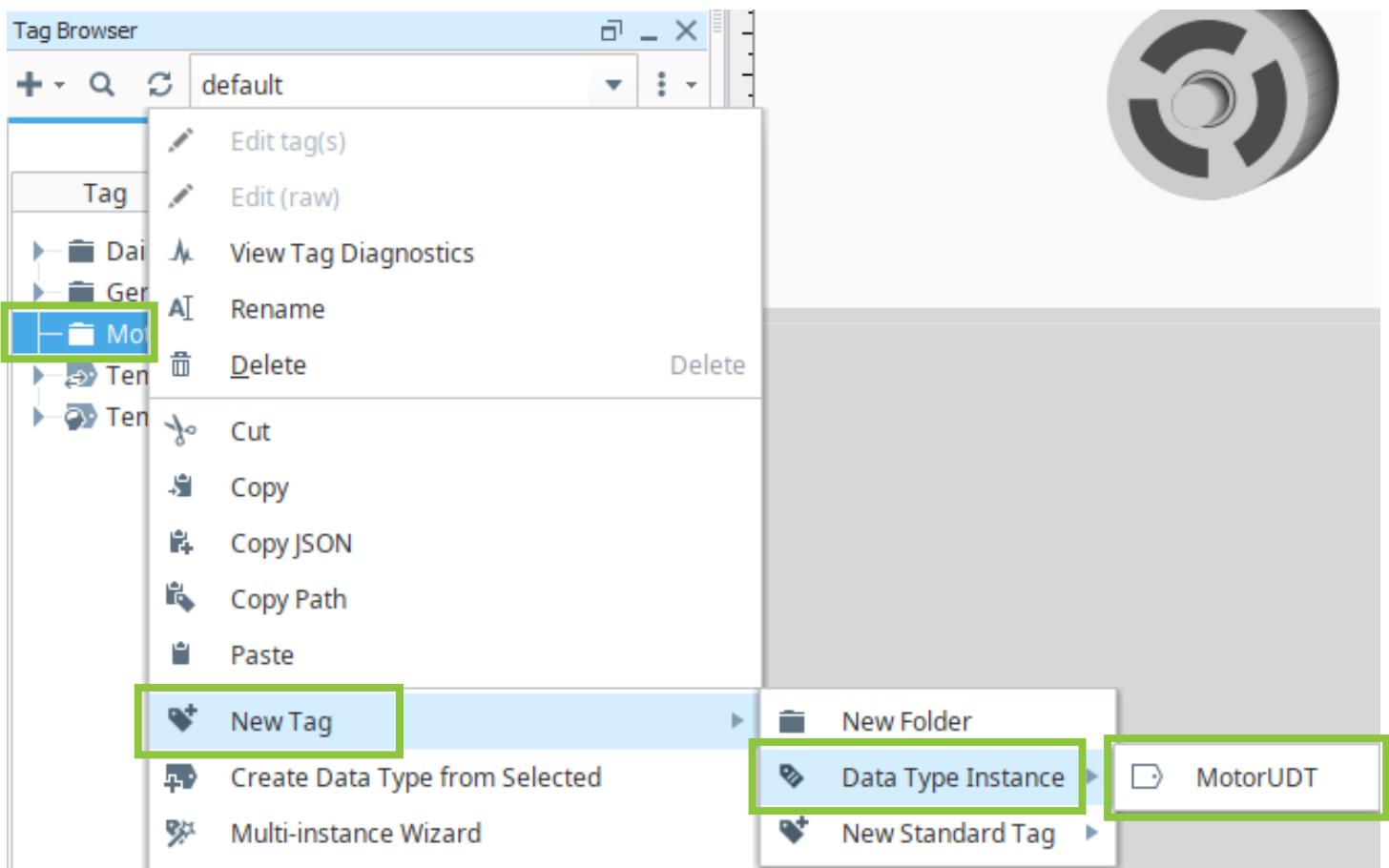
1. Click **Tags** in the Tag Browser.
2. Collapse any open folders.
3. Click  in the top left corner of the Tag Browser.
4. Click **New Folder**.
5. Enter **Motors**.

6. Click **OK**.

Create a Single Instance

You can create a single instance of a UDT in the Tag Browser. While this is useful for class demonstration purposes, it would be rarely used in the real world. After all, the entire purpose of using UDTs is to ease the creation and management of many Tags at once.

1. Right-click the **Motors** folder.
2. Select **New Tag**.
3. Select **Data Type Instance**.
4. Click **MotorUDT**.

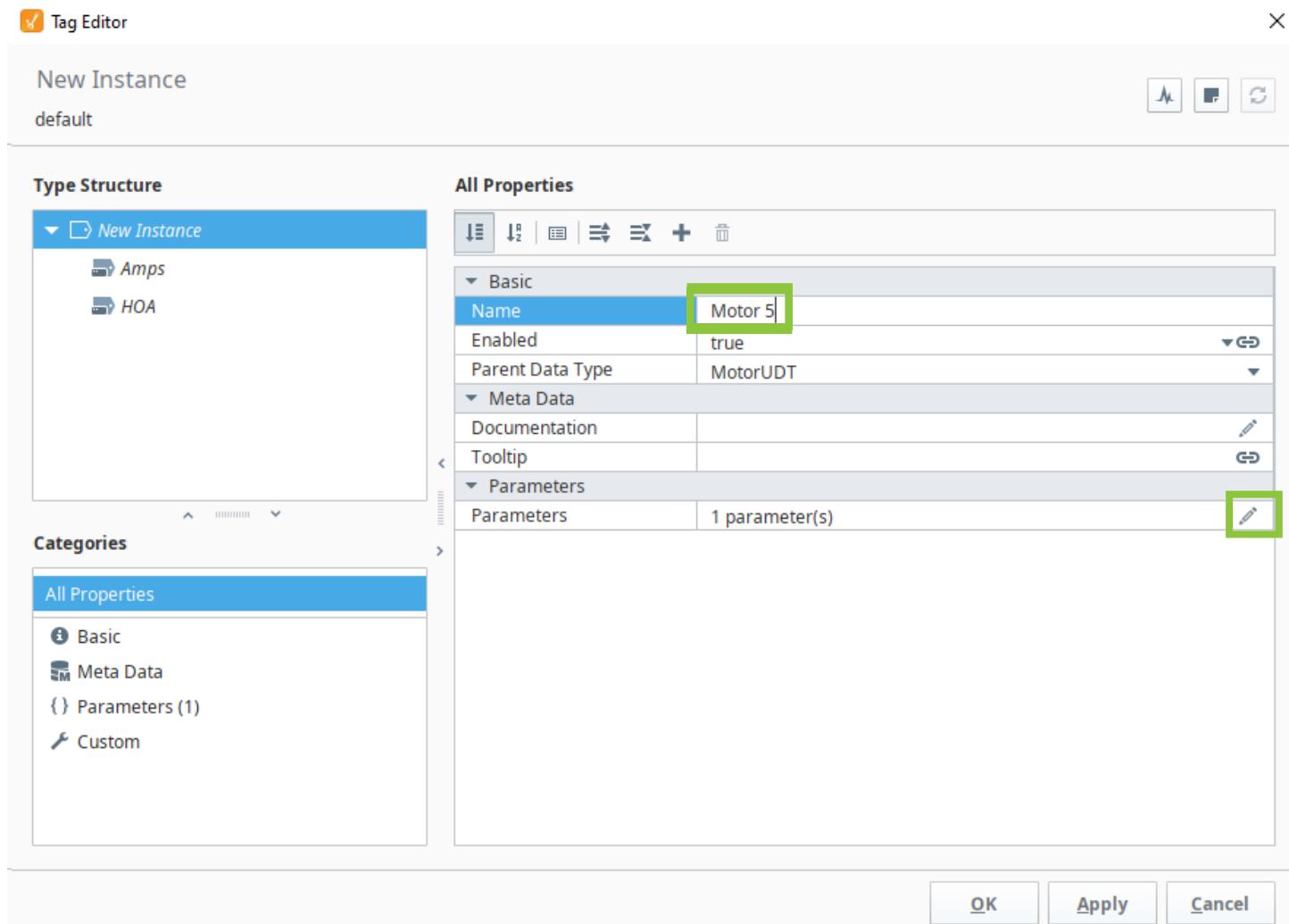


The Tag Editor opens.

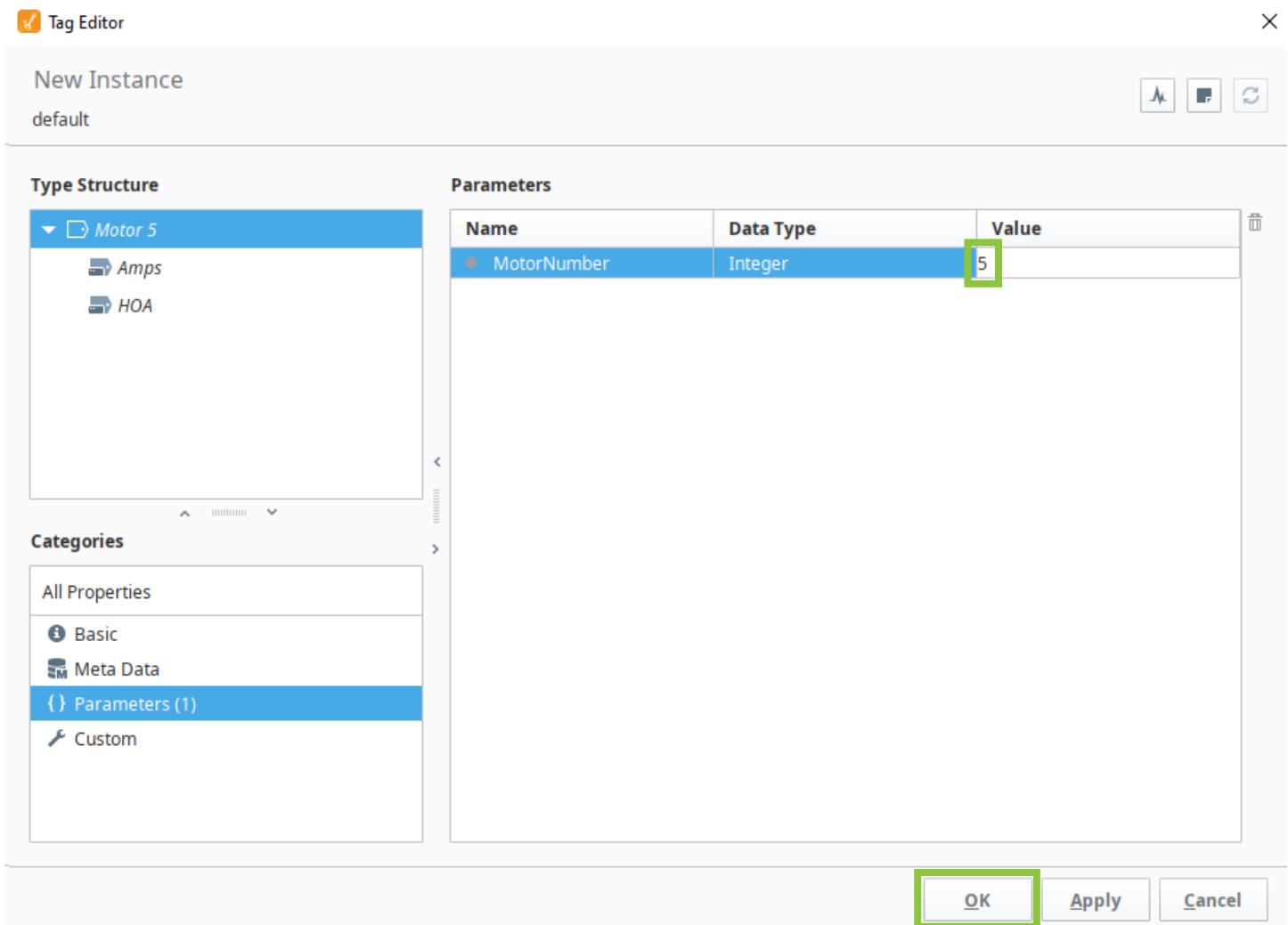
In order to create an instance of a UDT, we need to provide, at a minimum, two things: a unique name, and a value for each parameter.

Ch 9. Complex Tags (UDTs)

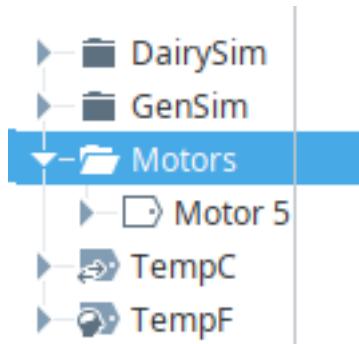
5. Enter **Motor 5** in the Name field. Be sure to include the space between the word and the number.
6. Click  to the right of Parameters.



7. Double-click the **Value** field.
8. Enter **5**.
9. Click **OK**.



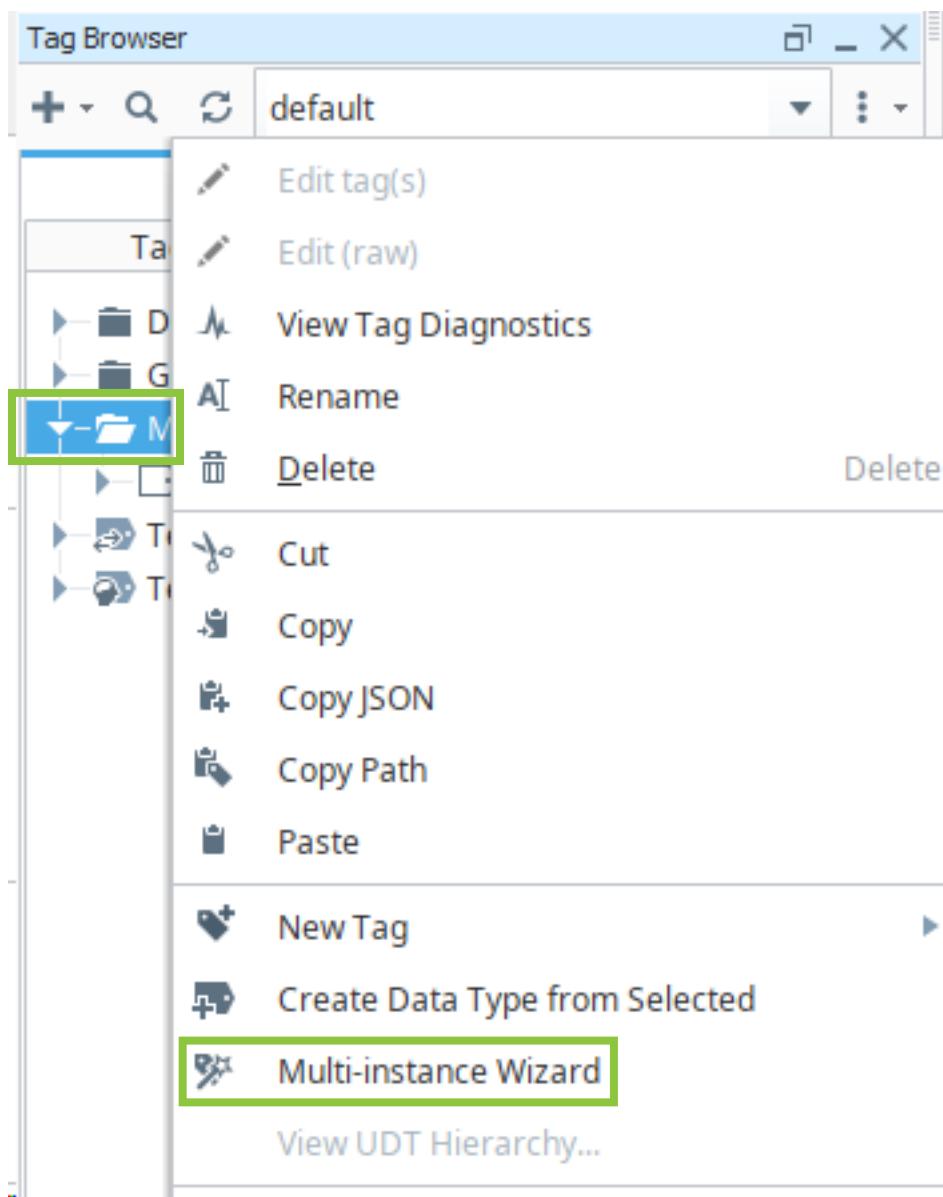
The instance is created. You may need to expand the Motors folder to see it.



Create Multiple Instances

More often than not, you will want to create many instances of a UDT. Thankfully, this is a simple process with the Multi-instance Wizard.

1. Right-click the **Motors** folder.
2. Click **Multi-instance Wizard**.

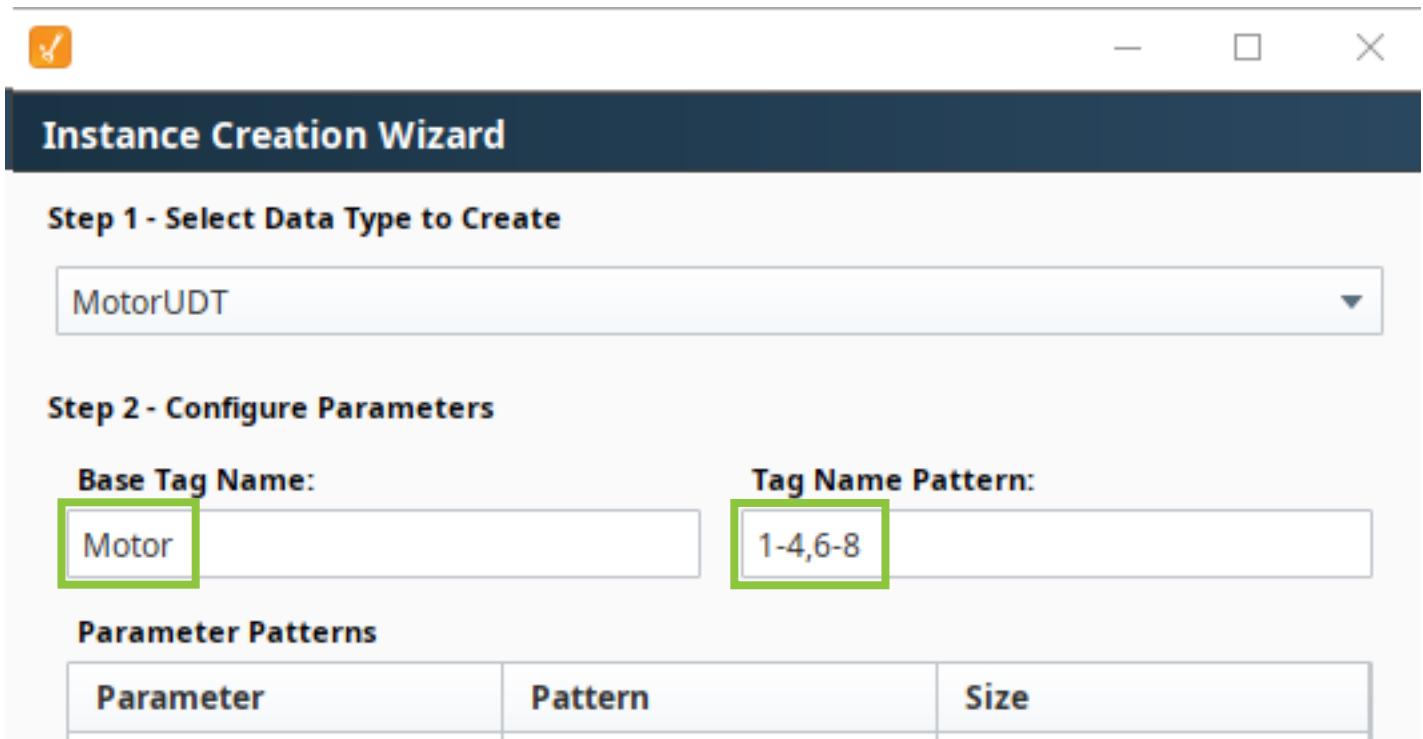


The Instance Creation Wizard opens.

The first step in the wizard is to select the UDT to create instances from, but in this case we only have a single UDT, so we can skip to the next step in the wizard.

3. Enter **Motor** in the Base Tag Name field.
4. Enter **a space** after the word.
5. Enter **1-4,6-8** in the Tag Name Pattern field.

Note: Make sure there is no space following the comma in the pattern.



There are a wide variety of Tag name patterns possible. In this case, we need the numbers 1-4 and 6-8, because we have already created Motor 5. If we had not created that individual instance, we could simply use 1-8 as the pattern.

Note that we have a simple naming scheme here, with a word followed by a number. More complex environments will require more complex naming schemes, and many options for the pattern are outlined in the blue box at the bottom of the wizard.

In addition to the naming pattern, we also need to create the pattern for the parameter. In our case, it is the same as the naming pattern, but it will not always necessarily be the same.

6. Double-click the **Parameter** field.
7. Enter **1-4,6-8**.

Note: Once again, ensure that there is no space in the pattern.

8. Press **Enter**.

Parameter Patterns

Parameter	Pattern	Size
MotorNumber	1-4,6-8	7

Tags to create: 7

The Size and Tags to create will both update to show that the wizard will create 7 Tags.

9. Click **Preview**.

Verify that the Tags to be created have the correct names and parameter numbers. If they do not, click **Back** and make any necessary corrections.

Instance Creation Wizard

Tag Name	MotorNumber
Motor1	1
Motor2	2
Motor3	3
Motor4	4
Motor6	6
Motor7	7
Motor8	8

[Back](#) [OK](#) [Cancel](#)

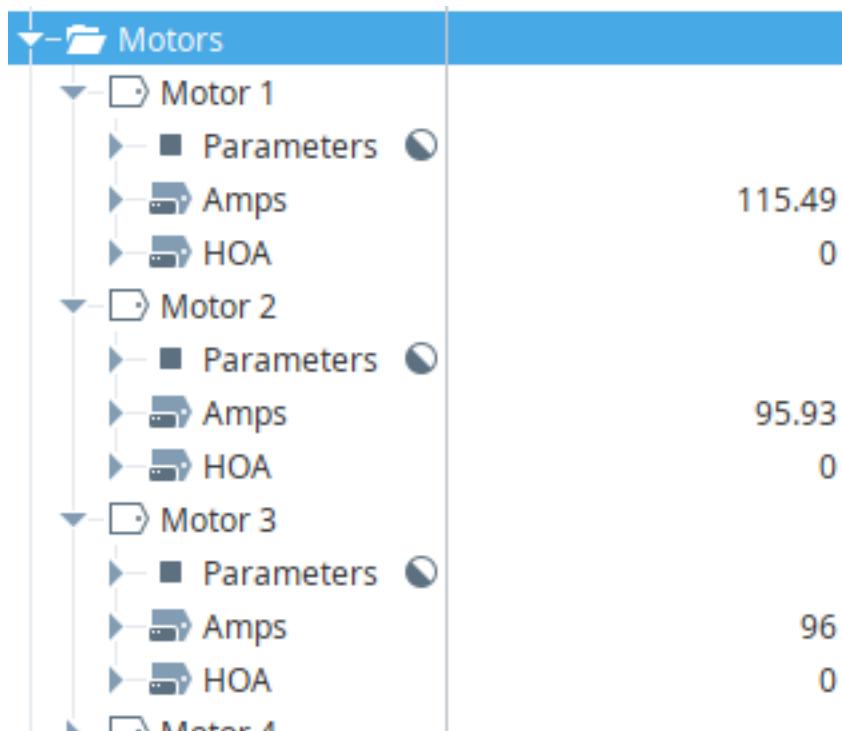
Best Practice: Always Use Preview

Once you click OK to have the wizard create the Tags, there is no simple way to fix a mistake in the naming convention or parameters. Therefore, it's highly recommended that you always use the Preview button to verify that you have everything configured correctly before closing the wizard.

10. Click **OK**.

The remaining Tags are created.

You can verify that the newly-created Tags are correct by expanding one, and then expanding the Amps or the HOA Tag, and viewing the OPC Item Path to see that the number in the path is being properly replaced by the parameter.



Editing a UDT

One of the primary benefits to using UDTs is that you can edit the UDT definition, and automatically have those changes applied to every instance.

To demonstrate this, we are going to add a Memory Tag to each Motor. We will actually use this Tag later in the course; for now, we just want to see that adding it to the UDT automatically adds it to each instance.

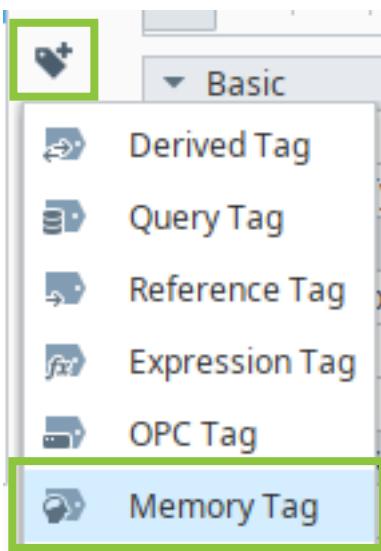
1. Click **UDT Definitions** in the Tag Browser.

2. Double-click **MotorUDT**.

The UDT Definition Editor opens.

3. Click  .

4. Click **Memory Tag**.



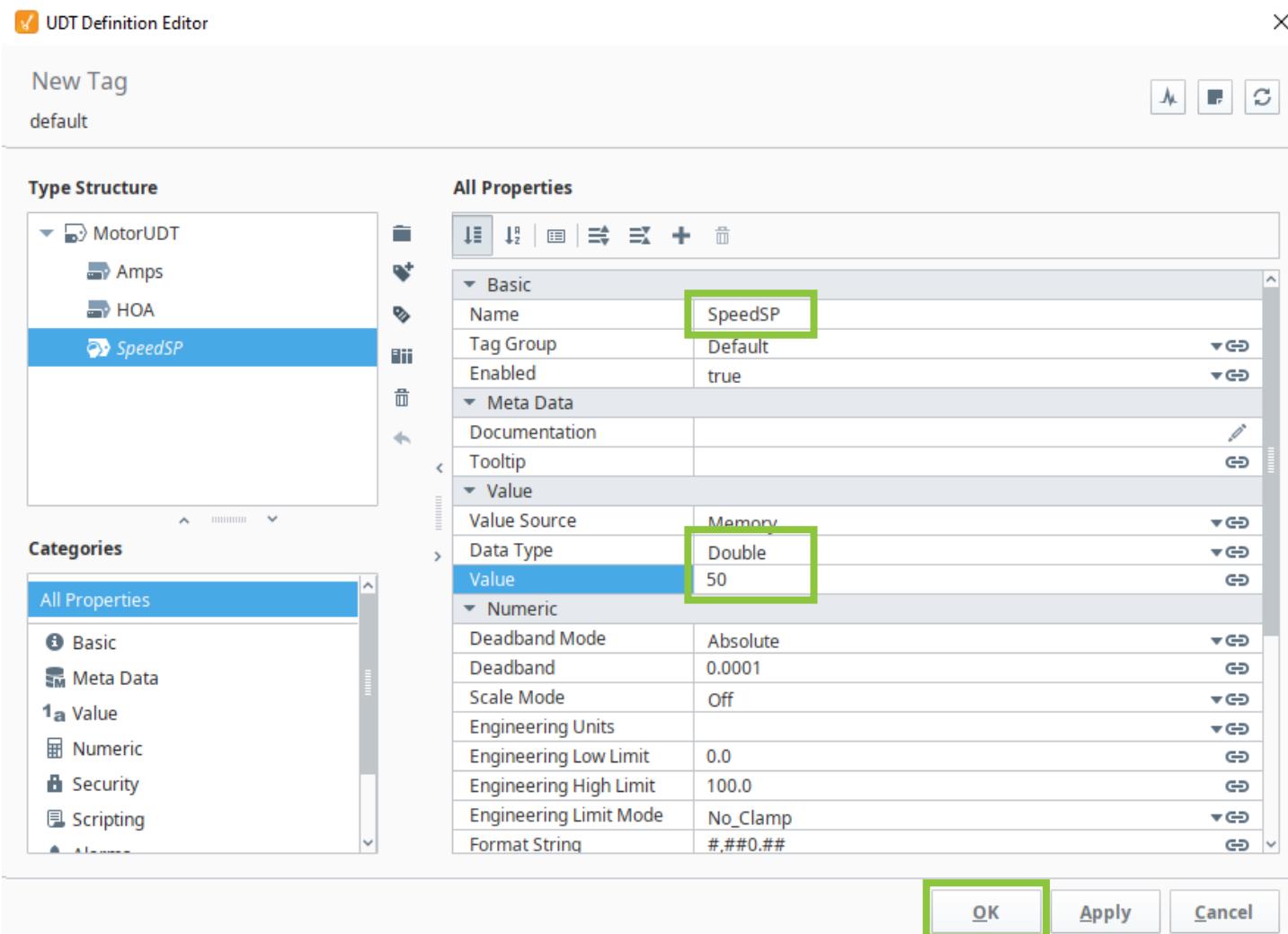
A new Tag is added to the UDT.

5. Enter **SpeedSP** in the Name field.

6. Select **Double** for the Data Type.

7. Enter **50** in the Value field.

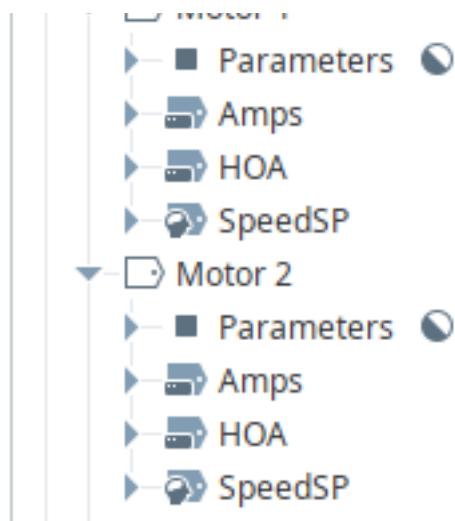
8. Click **OK**.



9. Click **Tags** in the Tag Browser.

10. Expand several of the **Motors**.

Each Tag now has a SpeedSP memory Tag.



When we created the TempF Memory Tag earlier, we noted that it was a global variable. Memory Tags in UDTs still function like variables, but now, each one is unique to its Tag instance—changing the value one will not impact any of the others.

Overrides

We all know that not every motor is exactly the same, so there is a way to make changes to individual Instances of a UDT. You can override any property of a UDT by going into the Tag Editor and changing any property or value. When you override a value, you will see a small green circle appear to the left of the property name. This lets you know that this property has a local override.

Once you have overridden a property on a UDT, further changes to that property on the UDT itself will not impact the overridden property on the instance, so this needs to be used with caution.

You will also notice an icon in the Tag Browser whenever a property is overridden. In fact, each of our motors already has an override, as the value of the parameter, by definition, needs to be unique to each instance.

Nesting UDTs and Inheritance

When creating new UDTs, you can make them as simple or complex as you want. In the example above, we have a simple UDT with just three Tags. You can also include other UDTs inside a UDT. For example, you could have a UDT for a conveyor where each conveyor has a motor attached to it. You can re-use your motor UDT as a part of the Conveyor. This way when you update the Motor UDT, all Conveyor UDTs will be automatically updated.

UDTs can also have parent-child relationships. You might have several types of motors, where one is simple like the example above, and one is a variable frequency drive that has our simple motor as a parent and adds a few other Tags that control the motor speed. This way when you update the simple motor UDT, all child motor UDTs will be updated automatically.

Templates

Templates are a simple but a very powerful feature in Ignition that you can use in Vision. HMI and SCADA systems typically have a lot of repetition in their screens. You might use a group of the same components over and over within your project to represent different motors. The data driving each set of graphics is different, but the graphics themselves are copies of each other.

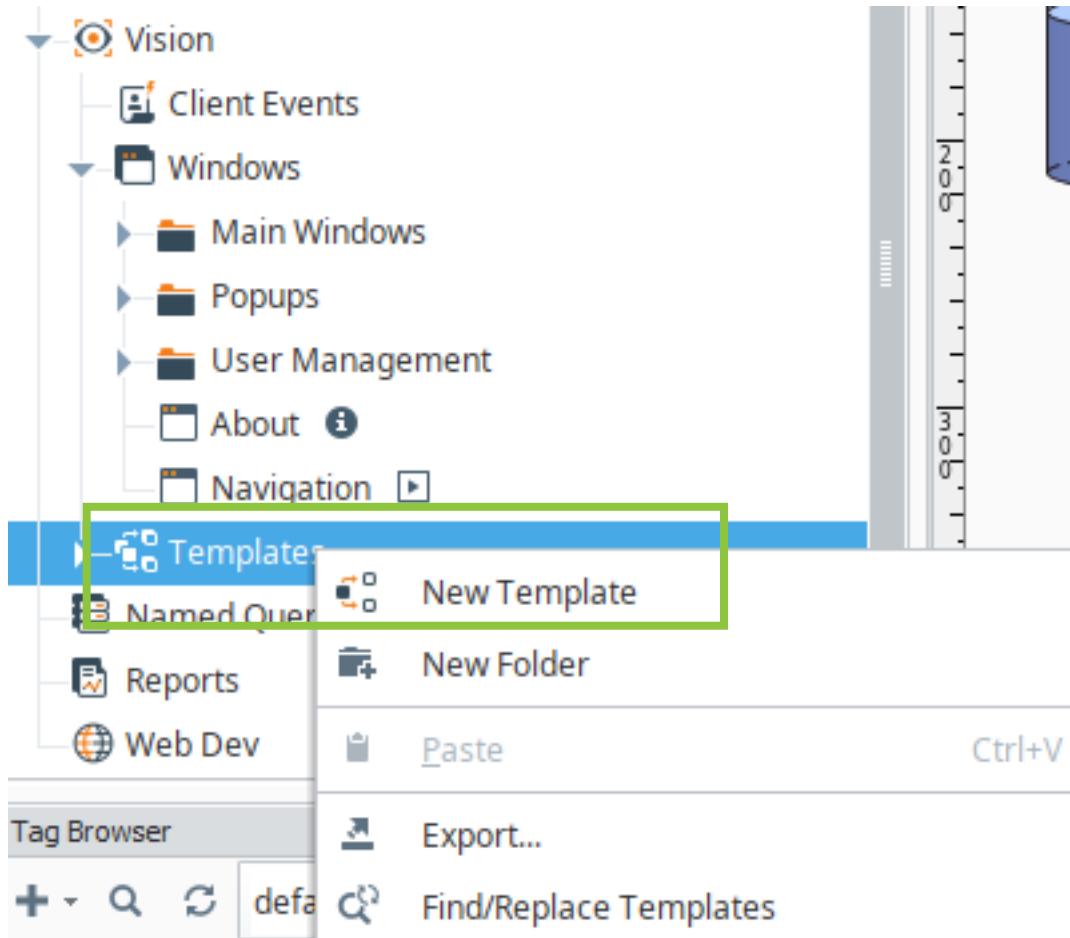
Without Templates, the only way to do this is to copy-and-paste the components each time you want to represent a motor. This is simple, and it works, but it can cause major headaches and time consuming corrections later on. If you ever want to make a change to how motors are represented, you're stuck making the change to each copy of the group. Templates solve all of these issues.

Create a Template

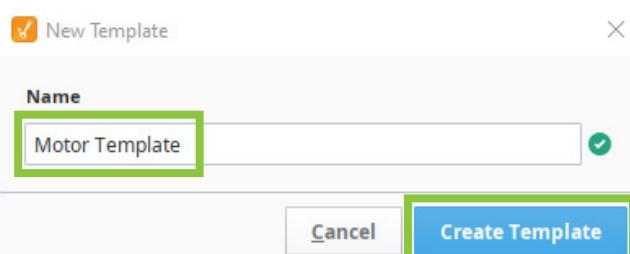
A Template is created in much the same way as a Vision window: you add components, arrange them as necessary, and bind them to each other or other data sources.

To create a Template, follow these steps:

1. Right-click **Templates** in the Project Browser.
2. Click **New Template**.



3. Enter **Motor Template** in the Name field.
4. Click **Create Template**.



A blank Template is created.

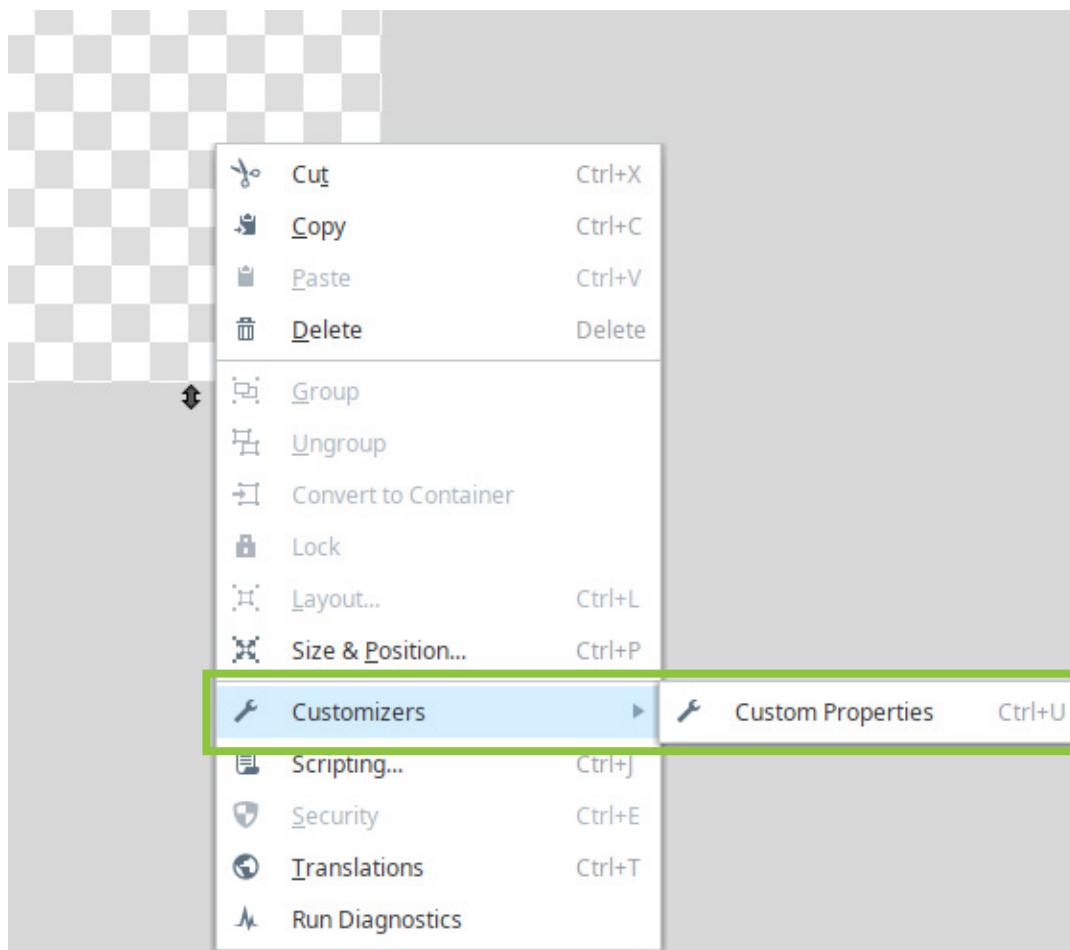
Note: The Template has a checkerboard pattern as its background because by default, Templates use a transparent background so that they can be placed on any window and adopt that window's background. The Template does, however, have a Background Color property that can be changed to any other color if you wish.

Template Parameters

The primary use of Templates is to create a generic version of an interface, and then create instances that will show specific instances. In much the same way that a UDT uses parameters to represent variables in instances, Templates have parameters that can be passed to them to change specific elements to reflect differences in instances.

To create a parameter for the Template, follow these steps:

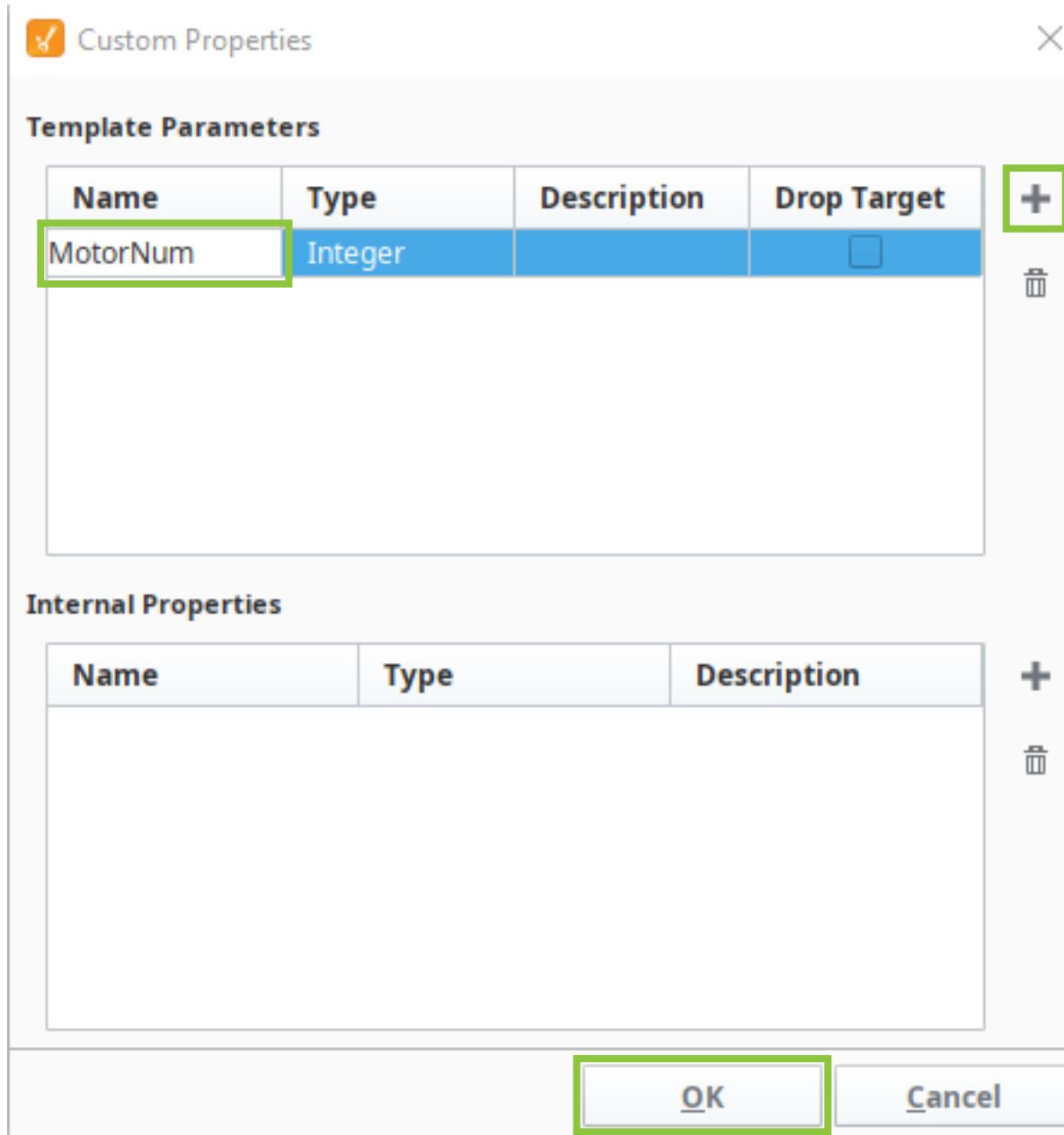
1. Right-click in the **background** of the Template.
2. Select **Customizers**.
3. Click **Custom Properties**.



4. Click .

5. Enter **MotorNum** in the Name field.

6. Click **OK**.

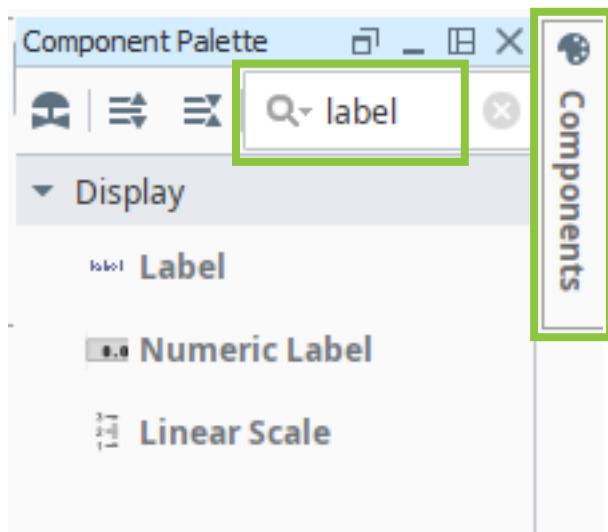


Note: Even though the Template's MotorNum parameter and the UDT's MotorNumber parameter represent the same thing—the number of the motor—they are completely unrelated. We are therefore giving them slightly different names, but that is to make clear that they are not the same. It is perfectly acceptable to use the same name for both.

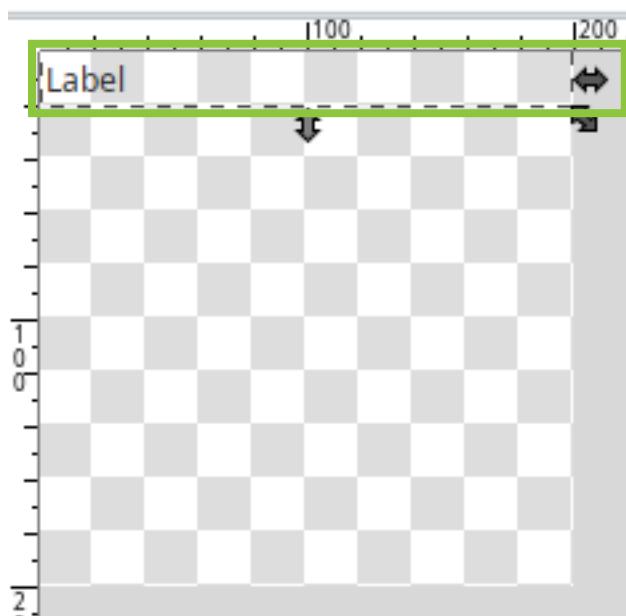
Adding Components to Templates

You can add components to your Template in exactly the same way you add them to windows.

1. Expand the **Components Palette**.
2. Enter **Label** in the Filter box.

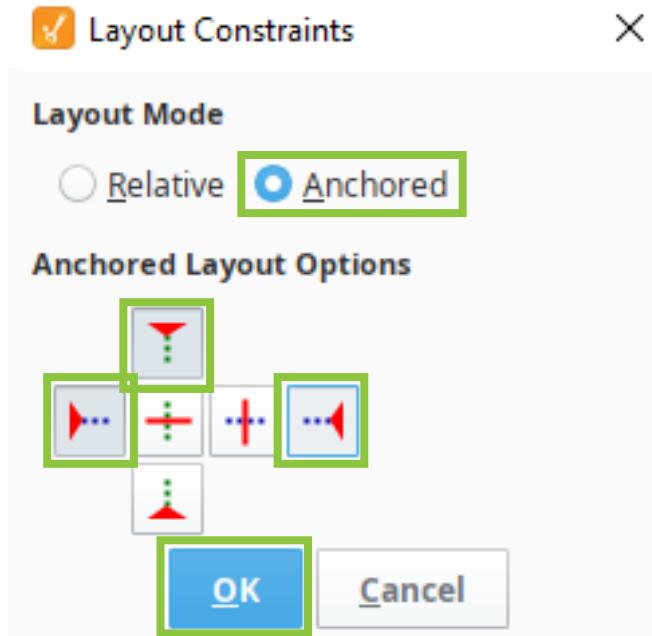


3. Drag a **Label** to the top left corner of the Template.
4. **Resize** the Label to the same width as the Template.



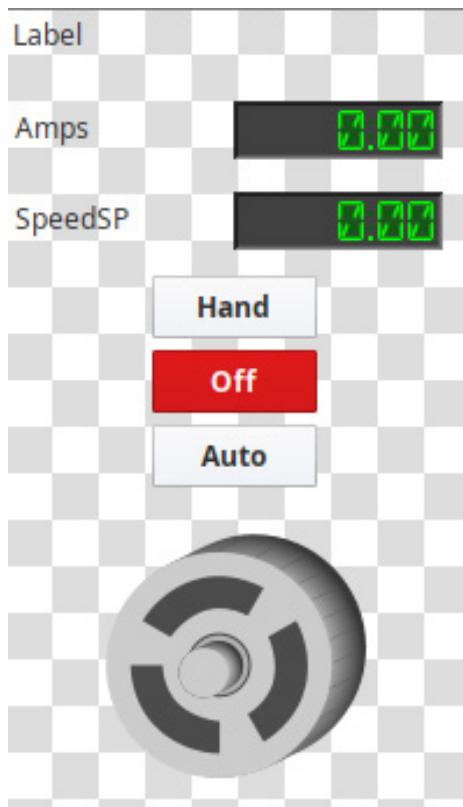
5. Right-click the **Label**.

6. Select **Layout**.
7. Select **Anchored**.
8. Click the **top**, **left** and **right** anchor buttons.
9. Click **OK**.



10. Expand the **Components Palette**.
11. Drag a **Label** to the Template.
12. Position the new Label near the left edge of the Template, below the previous Label.
13. Apply the same **layout** settings as above.
14. Expand the **Components Palette**.
15. Click to clear the text from the Filter box.
16. Enter **LED**.
17. Drag an **LED Display** to the Template.
18. Position the **LED** to the right of the second Label.
19. Apply the same **layout** settings as above.
20. Select both the Label and the LED.
21. Press the **Ctrl** key.

22. Drag the components down to duplicate them.
23. Enter **Amps** in the **Text** property of the first Label.
24. Enter **SpeedSP** in the **Text** property of the second Label.
25. Expand the **Components Palette**.
26. Click  to clear the text from the Filter box.
27. Enter **Multi**.
28. Drag a **Multi-state Button** to the Template.
29. Position the button below the existing components.
30. Apply the same **layout** settings as above.
31. Click **Tools**.
32. Click **Symbol Factory**.
33. Click **Enhanced**.
34. Click **Motors**.
35. Drag **Simple motor 4** to the Template.
36. Position the motor below the other components. It may be necessary to resize the Template.



Create an Expression With a Template Parameter

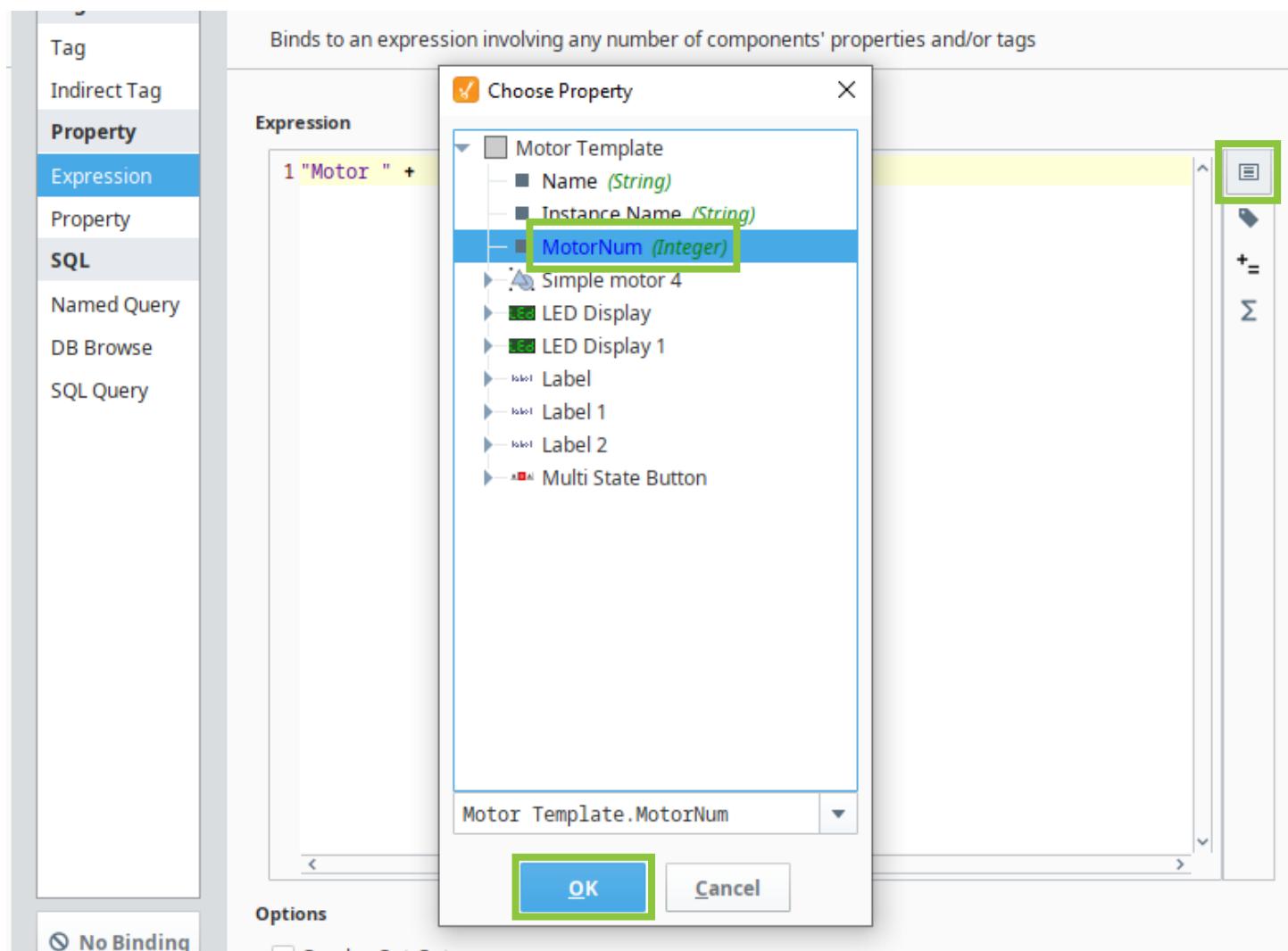
We have already seen how we can use expressions to create dynamic bindings on components. We can also use the Template parameter as a variable in the expression.

1. Click the first label on the Template.
2. Click  for the label's Text property.
3. Click **Expression**.
4. Enter the following:

```
"Motor " +
```

Be sure to include the space between the word and the closing quotes.

5. Click .
6. Click **MotorNum**.
7. Click **OK**.



8. Confirm that your code looks like this:

```
"Motor " + {Motor Template.MotorNum}
```

9. Click **OK**.

Indirect Tag Bindings

We have already seen how we can bind Components to Tags, but Templates require a slightly different method. The bindings we have used so far have been attached to a specific Tag. But now, we need a more generic binding that can use the Template's parameter as part of the binding.

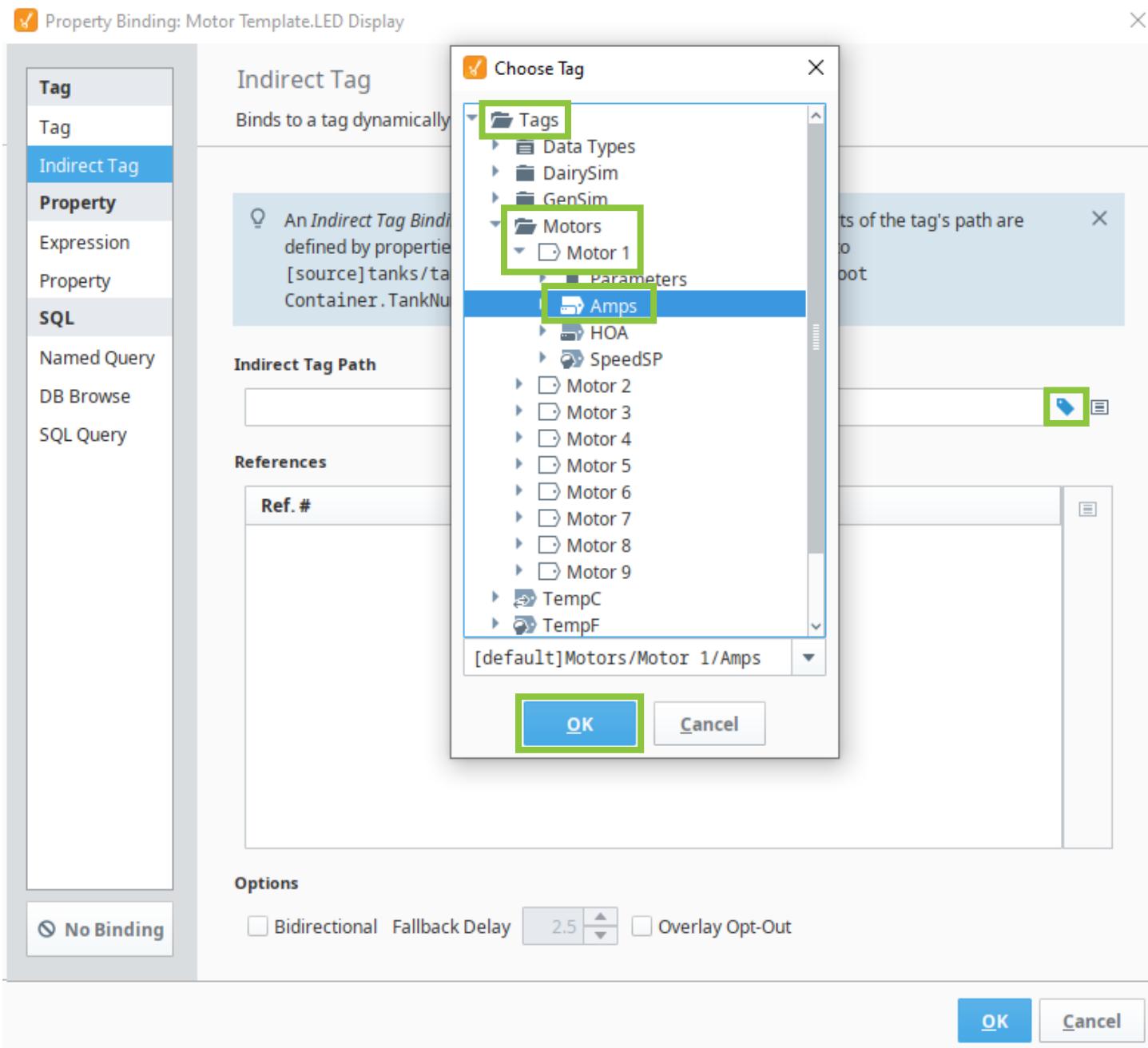
1. Click the top **LED display**.
2. Click  on the LED's **Value** property.
3. Click **Indirect Tag**.

To create the indirect binding, we first need a sample Tag path. We can do this by using the Insert Tag feature to find a Tag from one of the instances of our UDT. Then, we can modify the one part of that path that matches the Template parameter value by using the Insert property value feature.

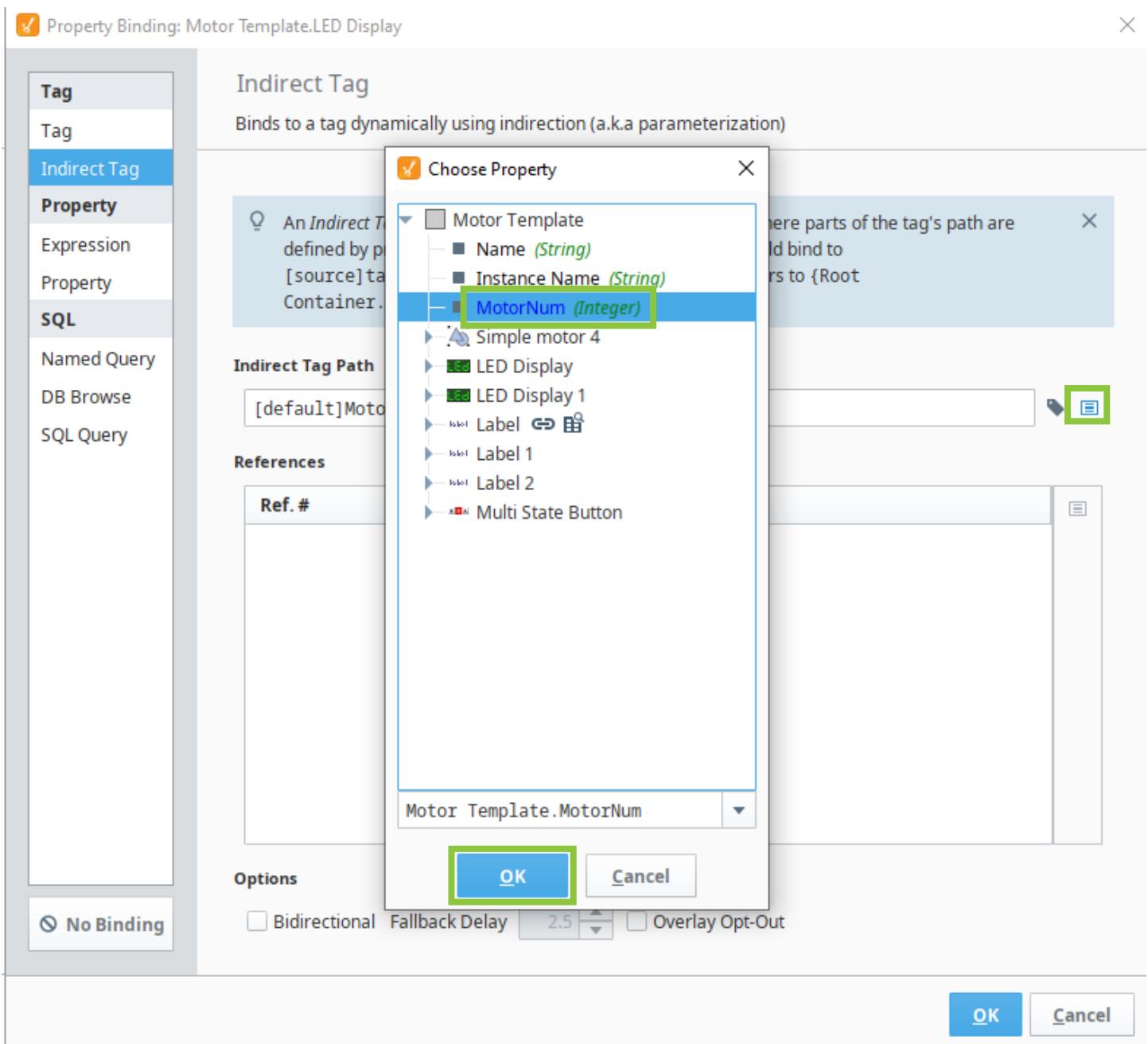
1. Click .
2. Expand **Motors**.
3. Expand **Motor 1**.

Note: You can select any of the motors.

4. Click **Amps**.
5. Click **OK**.



6. Select the **1** in the path.
7. Press **Delete**.
8. Click .
9. Click **MotorNum**.
10. Click **OK**.



11. Ensure your final Indirect Tag Path looks like this:

Indirect Tag Path

```
[default]Motors/Motor {1}/Amps
```

References

Ref. #	Property Path
1	Motor Template.MotorNum

You can now test this by clicking in a blank area of the Template's background and confirming the current value of the **MotorNum** parameter.

Note: There is no way to set an initial default value for the parameter, so the first time you examine it, the value will likely be 0. This will cause your binding to appear to not work, as there is no Motor 0. Simply changing the parameter value to a number between 1 and 8 will get the LED display to start working.

Compare the value of the **Amps** Tag in the Tag Browser for the Motor that matches your Template parameter with the value being displayed in the LED. Then, change the Template parameter and check to make sure the LED changes to display that Motor's Amps.

Now, we can repeat this process with the other LED.

1. Click the second **LED** display.
2. Click  for the LED's Value property.
3. Click **Indirect Tag**.
4. Click 
5. Expand **Motors**.
6. Expand any **Motor**.
7. Click **SpeedSP**.
8. Select the **number** in the path.
9. Press **Delete**.
10. Click .
11. Click **MotorNum**.
12. Click **OK**.
13. Click **OK**.

Note: Remember that as a Memory Tag, SpeedSP has a single, static value, so it will not constantly update in the LED.

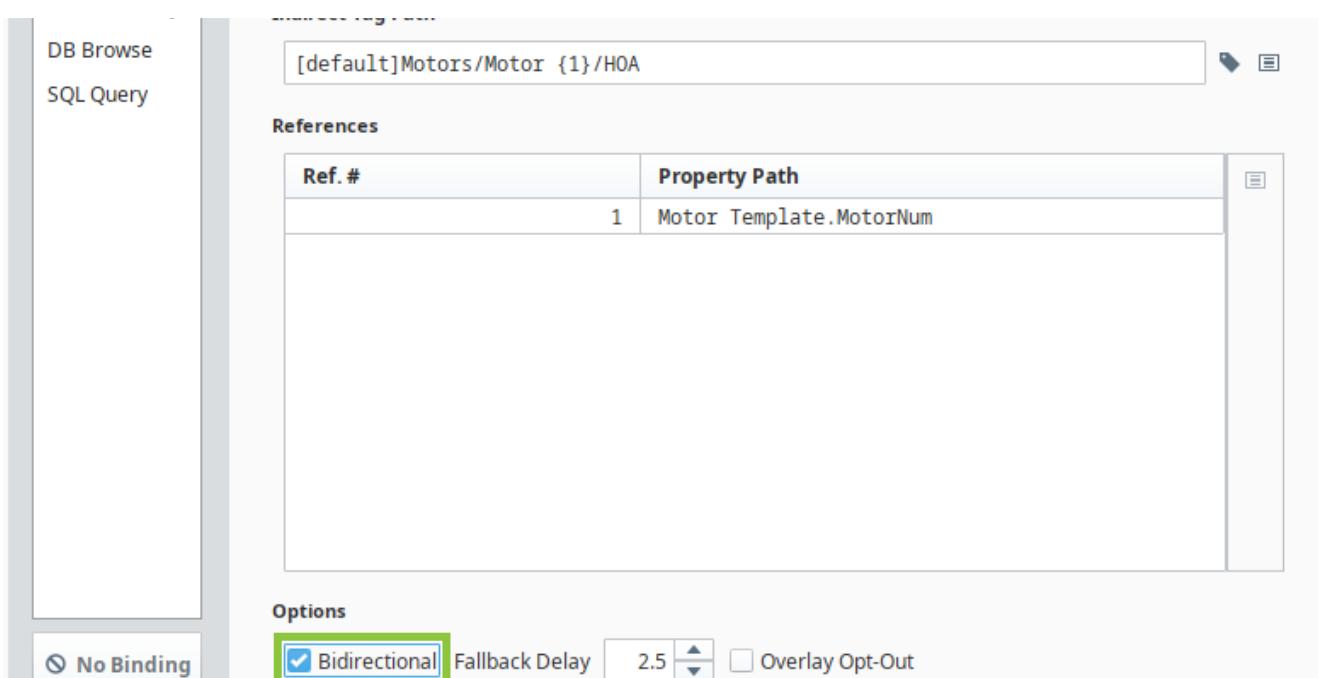
Indirect Tags With a Multi-state Button

We want to use the Multi-state Button to both view and control the value of the HOA Tag. The button has two distinct properties for this. The **Control Value** is a writable

property that can change the state of the Tag, while the **Indicator Value** is a read-only property that reads the value of the Tag and changes the button's state accordingly. In order to set up the button, we will need to set Indirect Tag bindings on both properties.

1. Click the **Multi-state Button**.
2. Click  for the LED's Value property.
3. Click **Indirect Tag**.
4. Click .
5. Expand **Motors**.
6. Expand any **Motor**.
7. Click **HOA**.
8. Select the **number** in the path.
9. Press **Delete**.
10. Click .
11. Click **MotorNum**.
12. Click **OK**.
13. Click **Bidirectional**.

Note: It is critical to check the Bidirectional checkbox. If you do not, the Control Value will not be able to write to the Tag.

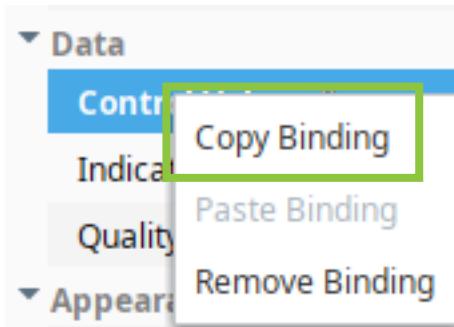


14. Click **OK**.

Copying Bindings

The Multi-state Button's **Indicator Value** needs the exact same binding as its **Control Value**. Rather than repeat all of the steps above, we can simply copy and paste the Control Value binding onto the Indicator Value.

1. Right-click **Control Value**.
2. Click **Copy Binding**.



3. Right-click **Indicator Value**.
4. Click **Paste Binding**.

That was simple, but it does raise a question. The Control Value's binding includes the Bidirectional setting, which allows the property to write to the Tag. We need that for Control Value, but not Indicator Value. However, when we copied the binding, we copied the entire binding, including that Bidirectional setting. In this particular case, it doesn't matter. The Indicator Value property of a Multi-state Button cannot write to a Tag, so the Bidirectional setting in this case will not do anything even if it is set. But you should be cautious when copying and pasting bindings to and from other components, as some settings like this will cause issues.

Adding Template Instances to Windows

Now that we have the Template created, we can use it in Vision Windows.

Once placed in a Window, a Template behaves exactly like a Component—it can be freely moved and resized, it is subject to layout behaviors, and it will have its own set of properties.

Create a New Vision Window

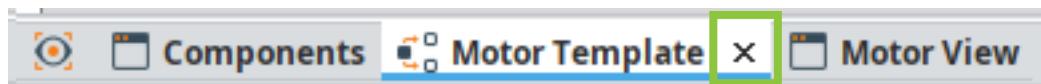
We need a window to add our Template to.

1. Expand **Windows**.
2. Expand **Main Windows**.
3. Right-click **Empty**.
4. Click **Duplicate**.
5. Right-click **Empty(1)**.
6. Click **Rename**.
7. Enter **Motor View**.
8. Click **OK**.
9. Double-click **Empty** on the header.
10. Enter **Motor View**.
11. If desired, change the icon. See “Changing the Header Icon” on page 81.

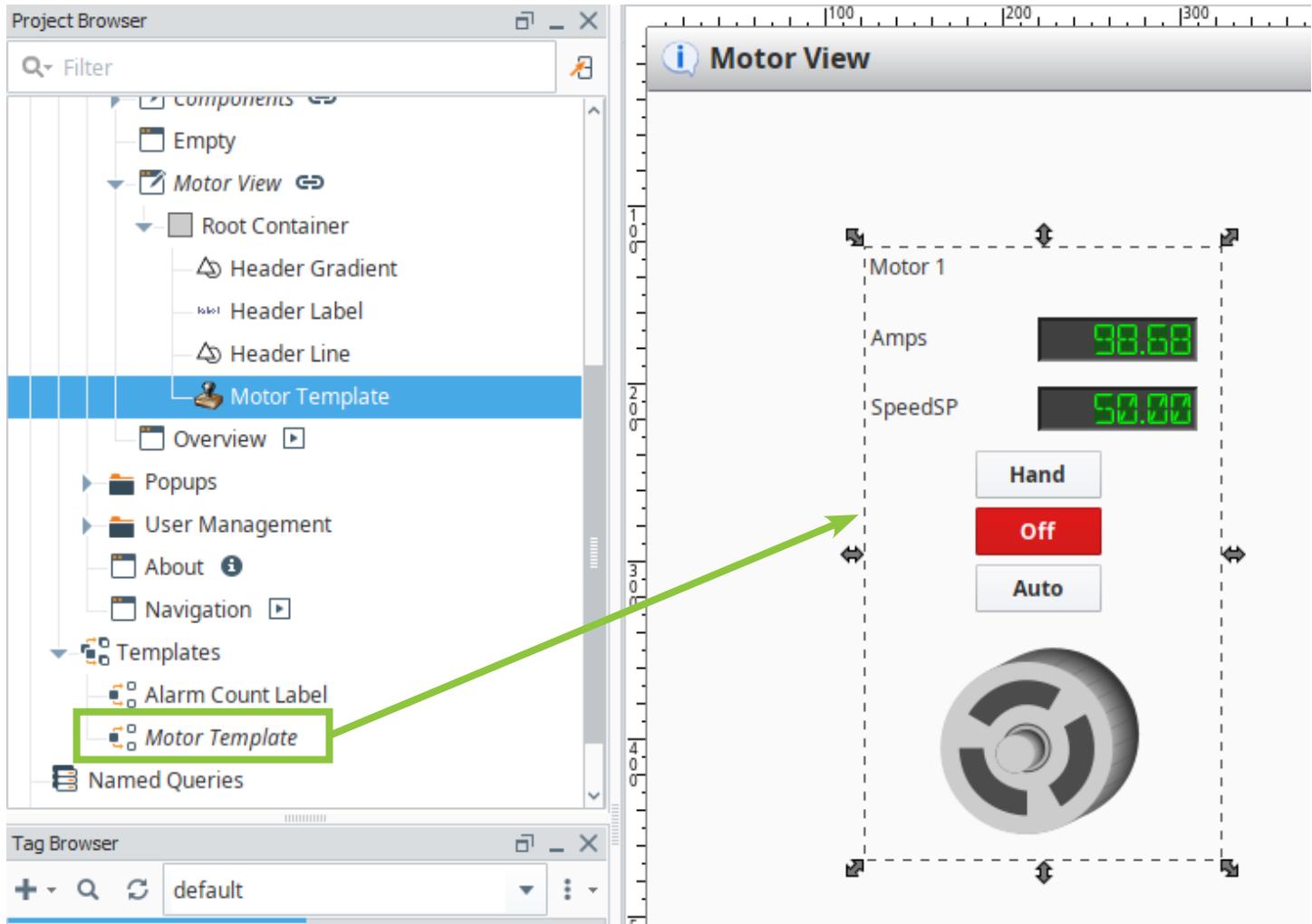
Add a Template Instance

So far, we have added Components to Windows by dragging them from the Components Palette. So, it might be reasonable to expect that we could drag a Template from the Project Browser to add it to a Window. And, we can, but there is a very big caveat: the Template itself cannot be open in the Designer. If it is, as soon as we try to click it in the Project Browser, Designer will switch to the Template, and attempting to drag it will end up dragging the Template into itself, which will result in an error.

1. Click **File**.
2. Click **Save All**.
3. Click **x** on the Template’s tab at the bottom of the Designer.



4. Click **Yes** to save the Template.
5. Drag **Motor Template** in the Project Browser to the Window.



Set the Template Parameter

Now that we have an instance of the Template, we can direct it to display a particular Motor. By default, new instances will have whatever value the parameter is set to on the Template itself.

1. Click the **Template instance**.
2. Change the value of **MotorNum** on the Vision Property Editor.



The Template will now display the details about the new Motor.

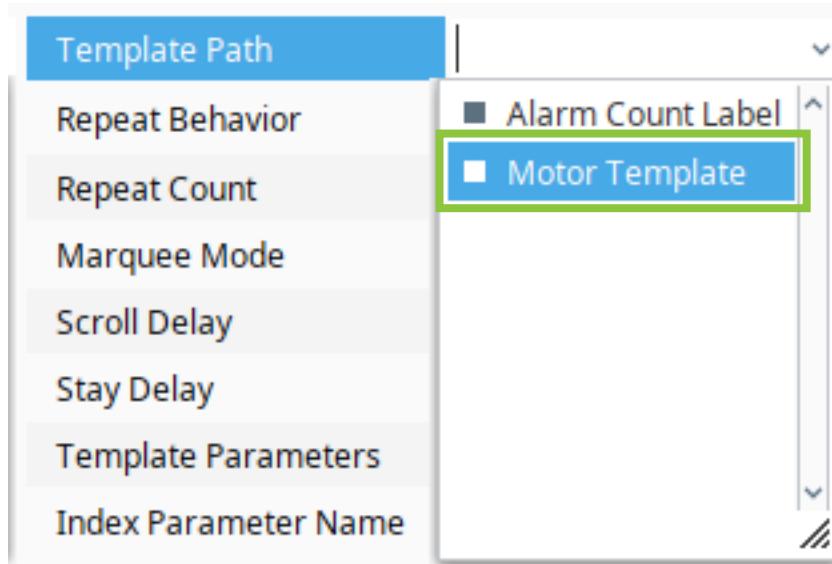
Add Additional Instances

You can have as many instances of a Template on a single Window as you need. You can add new instances by dragging them from the Project Browser, or by copying and pasting existing instances, or by holding Ctrl and dragging to duplicate instances. Each instance has its own **MotorNum** parameter that can be set to allow for a display of multiple Motors.

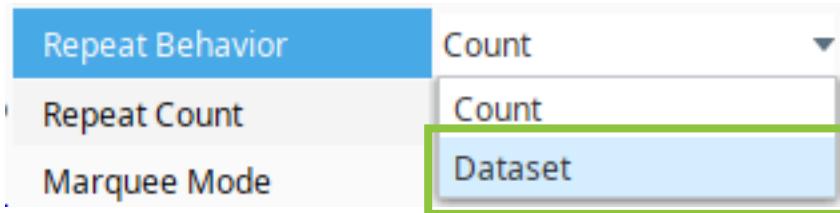
Use the Template Repeater

Another option for adding multiple instances of a Template to a Window is to use the Template Repeater component.

1. Expand the **Components Palette**.
2. Click  to clear the current filter.
3. Enter **Template**.
4. Drag the **Template Repeater** to the Window.
5. Resize the **Template Repeater**, and if necessary the Window, to create enough space to display 8 instances of your Motor Template in a grid **4** across and **2** high.
6. Select **Motor Template** from the **Template Path** property list.



7. Select **Dataset** from the **Repeat Behavior** property list.



8. Click  on the **Template Parameters** property.

The Dataset Editor opens.

The Template Repeater will populate itself with multiple instances of the Template set in the Template Path property, and is able to pass a unique value to the parameter in each instance. In order to do this, the Repeater needs a dataset, where each column is the name of a parameter in the Template, and each field is a parameter value. In our case, our Template only has a single parameter, so we need to create a dataset with a single column. More complex Templates might have multiple parameters and thus need more complex datasets.

9. Click .

10. Enter **MotorNum**.

Note: The name of the column must exactly match the name of the parameter. Pay close attention to capital letters and spacing.

11. Select **Integer** as the Type.

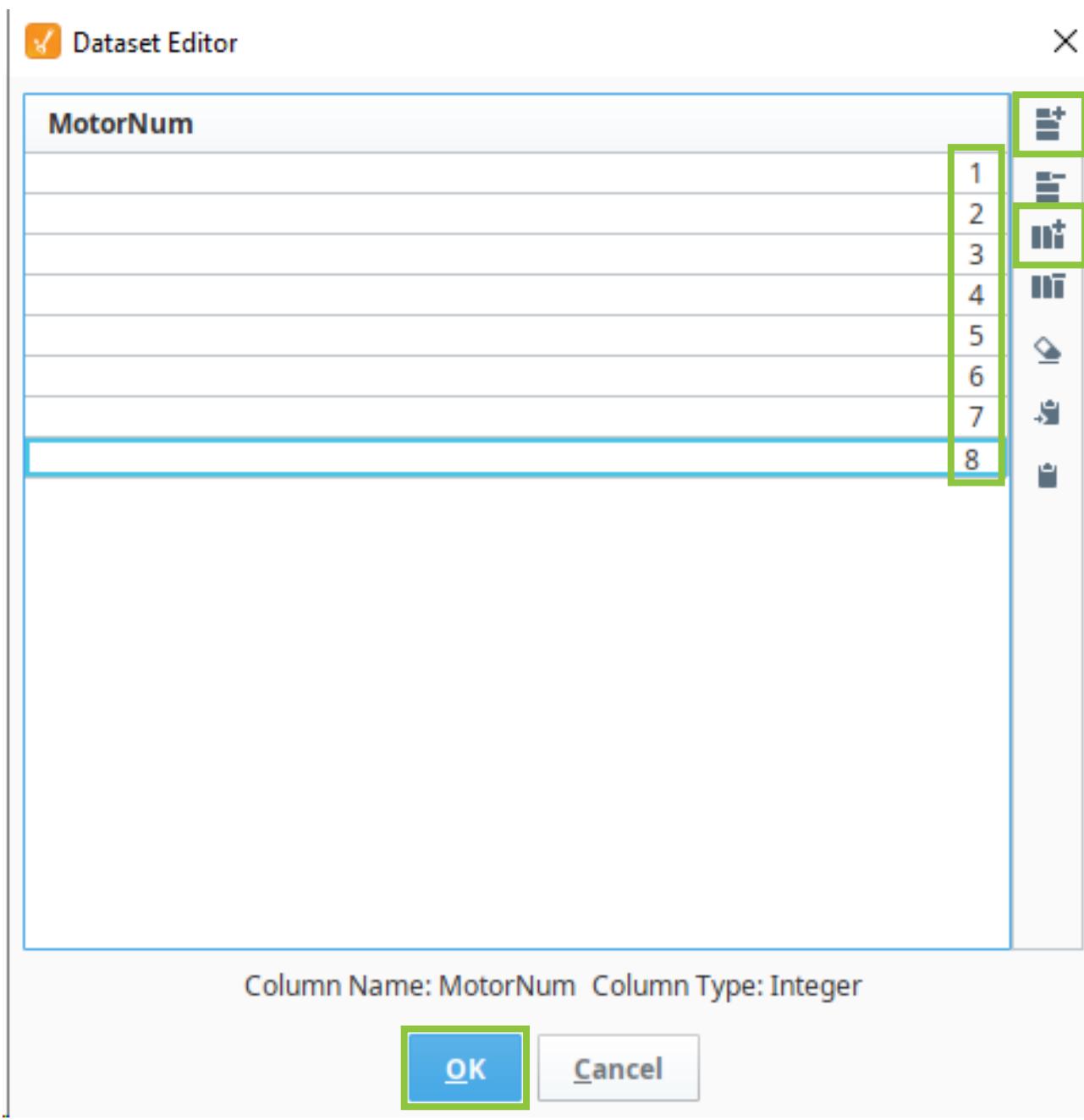
12. Click **Add Column**.

13. Click  8 times.

14. Set the values of the rows to **1** through **8**.

Note: You can enter the number of the first row, and then press Tab to move to the next.

15. Click **OK**.



The **Template Repeater** will now populate with eight instances of the Template.

We can tweak a few more settings to improve its appearance.

1. Select **Flow** from the **Layout Style** property list.
2. Select **Vertical** from the **Flow Direction** property list.

You can adjust the settings of the other properties in the **Appearance** section to further refine how the Template Repeater looks.

Best Practice: Datasets

After completing the last few steps, you might be asking yourself if manually creating the dataset for the Template Repeater is really necessary. Manually setting up the dataset with a single column and eight rows wasn't really difficult, but what if our Template had 5 parameters, and we needed to display 50 of them. Would we really need to set up a 5 by 50 dataset and then manually enter all 250 pieces of information?

Well, no. What we would do in a real-world environment would be have all of that information stored in a database. We could then use Ignition to query the database and return the information we need, and then simply bind the Template Repeater to the dataset. While that is beyond the scope of this class, it is covered in our advanced Database & Scripting class.

Use Parameters on a Symbol Factory Image

Our Template also includes a Symbol Factory image. It might not be obvious that there is anything bindable on the image, but in fact there are a host of properties we can potentially use to make the image dynamic.

One thing we can do is to add a color overlay to the image so that it can reflect the same colors as the Multi-State Button: red when the button (and thus the bound HOA Tag) is in the “Off” state, yellow when it is in the “Hand” state, and green when it is in the “Auto” state.

In order for this to apply to all instances of the Template, we want to set this up on the Template itself.

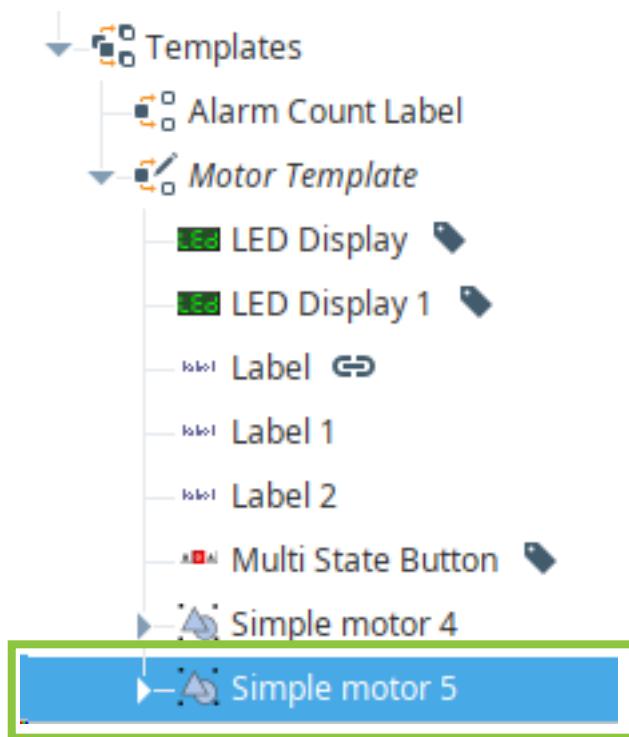
1. Double-click **Motor Template** in the Project Browser.
2. Expand **Simple motor 4** in the Project Browser.

You will see that the image is made up of three groups, which are in turn made up of a series of drawings.

In order to create the color overlay, we need a simplified version of this image that is a single shape, rather than a collection of many shapes. We can achieve this by creating a copy of the image, and then using the Shape tools to combine the elements of that copy into a simplified shape.

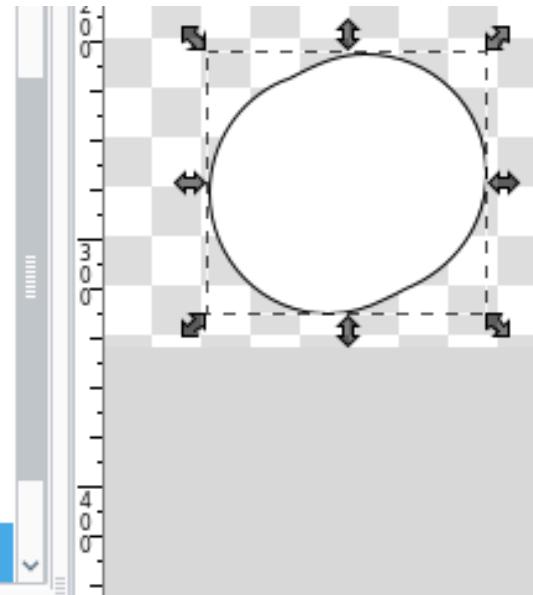
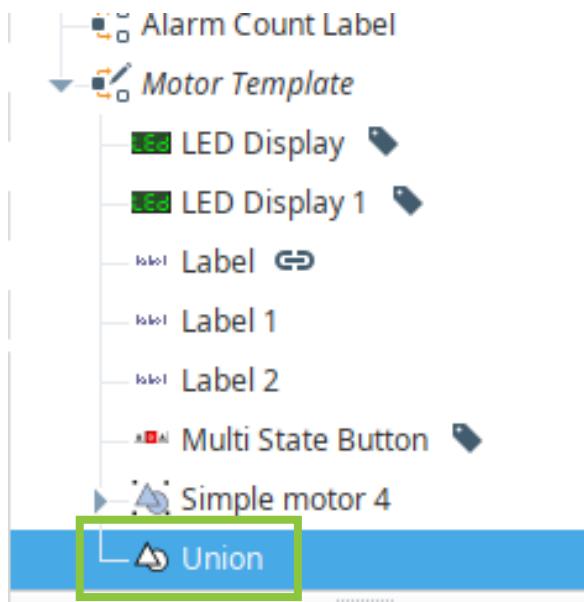
3. Click **Simple motor 4**.

4. Press **Ctrl+d** on your keyboard.
5. A copy, named **Simple motor 5**, is created.



Note: There is no other way to directly duplicate the image and place the duplicate exactly on top of the original other than using the keyboard shortcut.

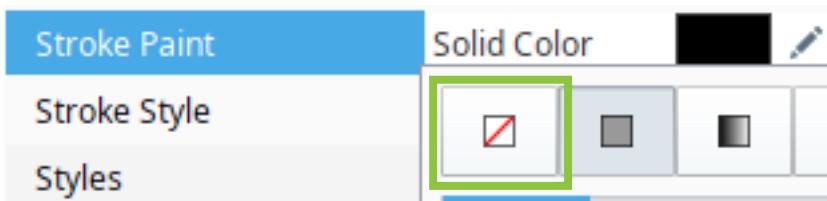
6. Click **Simple motor 5**.
7. Click  on the main toolbar.
8. The shape is simplified into a single object, now named **Union**.



Now that we have the simplified shape, we can use its **Fill Paint** property to create the color overlay we want. You can refer back to “Setting a Shape’s Fill and Stroke” on page 113 for details on using the color tools.

9. Click  on the Stroke Paint property.

10. Click **No paint**.



The Stroke is removed.

11. Click  on the **Fill Paint** property.

In order to achieve the effect we want, we will bind the shape’s **Fill Paint** to the **HOA** Tag using an Indirect Binding. However, we do have to ask: how will we convert the Tag’s three values—0, 1, 2—to a valid color value? In order for this to work, we will need to create the binding, and then add a color mapping to it, so that we can specify the color we want when the Tag is in each of its three states.

12. Click **Indirect Tag**.

13. Click .

14. Expand **Motors**.

15. Expand a **Motor**.

16. Click **HOA**.

17. Click **OK**.

18. Delete **the number** from the Tag path.

19. Click .

20. Click **MotorNum**.

21. Click **OK**.

 Property Binding: Motor Template.Union X

Indirect Tag

Binds to a tag dynamically using indirection (a.k.a parameterization)

? An *Indirect Tag Binding* lets you bind to a tag dynamically, where parts of the tag's path are defined by properties on your window. For instance, you could bind to [source]tanks/tank{1}_SP, and then define that {1} refers to {Root Container.TankNumber}

Indirect Tag Path

[default]Motors/Motor {1}/HOA ✖️ ☰

References

Ref. #	Property Path
1	Motor Template.MotorNum

Number-to-Color Translation

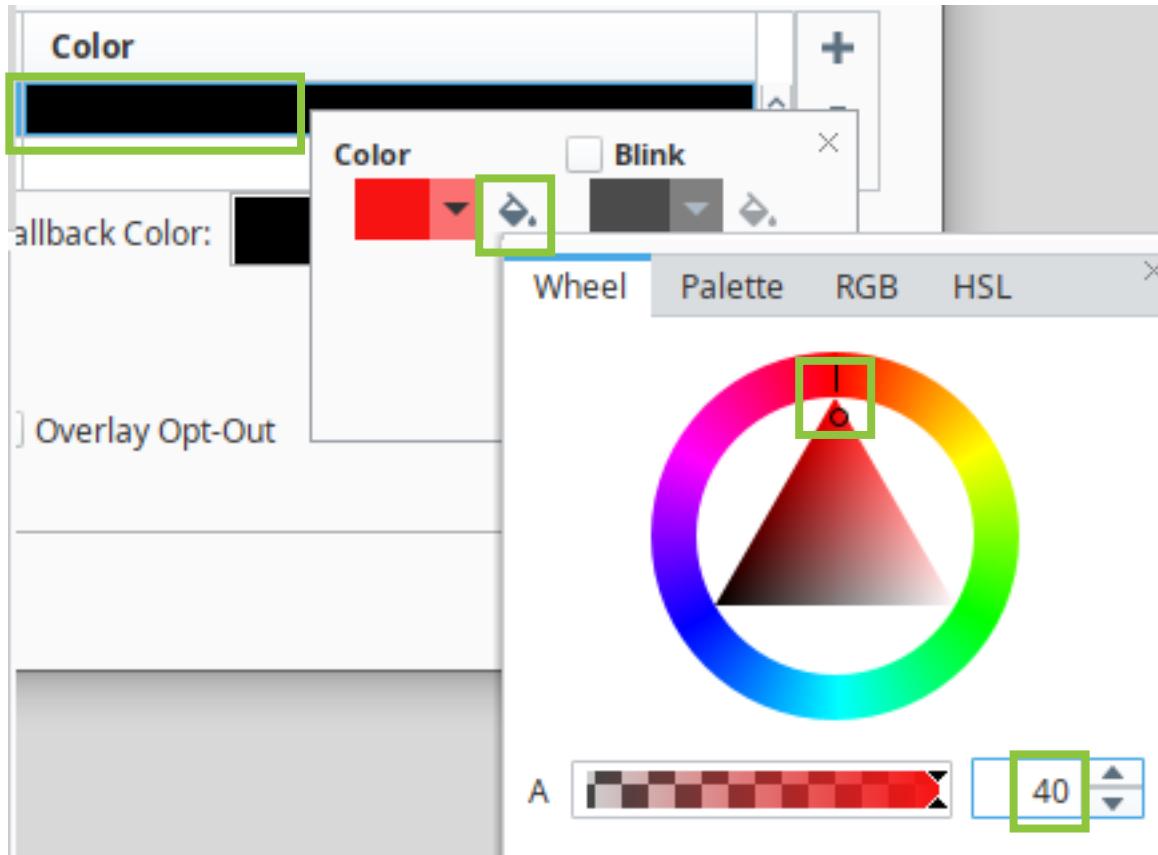
Value >=	Color	+
0		▲
1		▼

Options

No Binding Bidirectional Fallback Delay: Overlay Opt-Out

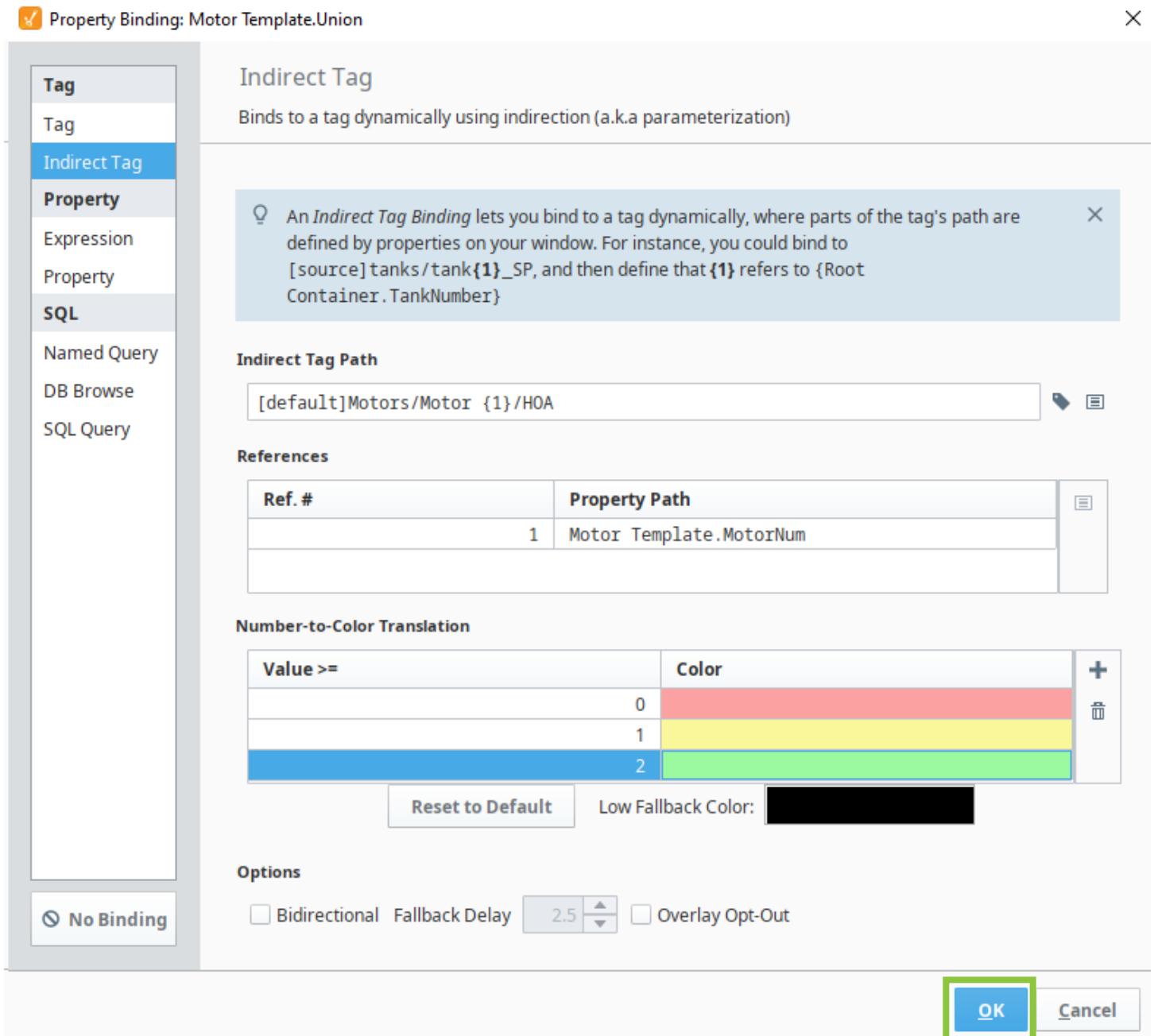
OK **Cancel**

22. Double-click the black bar in the first row of the **Number-to-Color Translation** table.
23. Click .
24. Click inside the triangle to select a shade of **red**.
25. Drag the **A** slider until the value reads **40**.



26. Click on the **Color popup**.
27. Click **OK**.
28. Double-click the **white bar** in the second row of the **Number-to-Color Translation** table.
29. Click .
30. Drag the line on the color wheel to **yellow**.
31. Click inside the **triangle** to select a shade of yellow.
32. Drag the **A** slider until the value reads **40**.
33. Click the **color popup**.
34. Click **OK**.
35. Click .
36. Double-click the white bar in the third row of the **Number-to-Color Translation** table.
37. Click .

38. Drag the line on the color wheel to a **green**.
39. Click inside the **triangle** to select a shade of **green**.
40. Drag the **A** slider until the value reads **40**.
41. Click the **Color popup**.
42. Click **OK**.
43. Click **OK**.



The binding is added to the shape.

You can see the result of this by returning to the **Motor View** window. You should see the image on each Template instance vary based on that Motor's HOA value. If necessary, you can go into Preview mode and use the Multi-State Button to alter the value of the Tag and see the image change to match.

Select Instances From a List

Another potential way we can display Template instances to our users is by showing a single instance and then allowing operators to select an instance from a list. This is going to be a better option for displays with limited screen space or limited computing resources, as the display only needs to ever show a single instance at a time.

Add a List

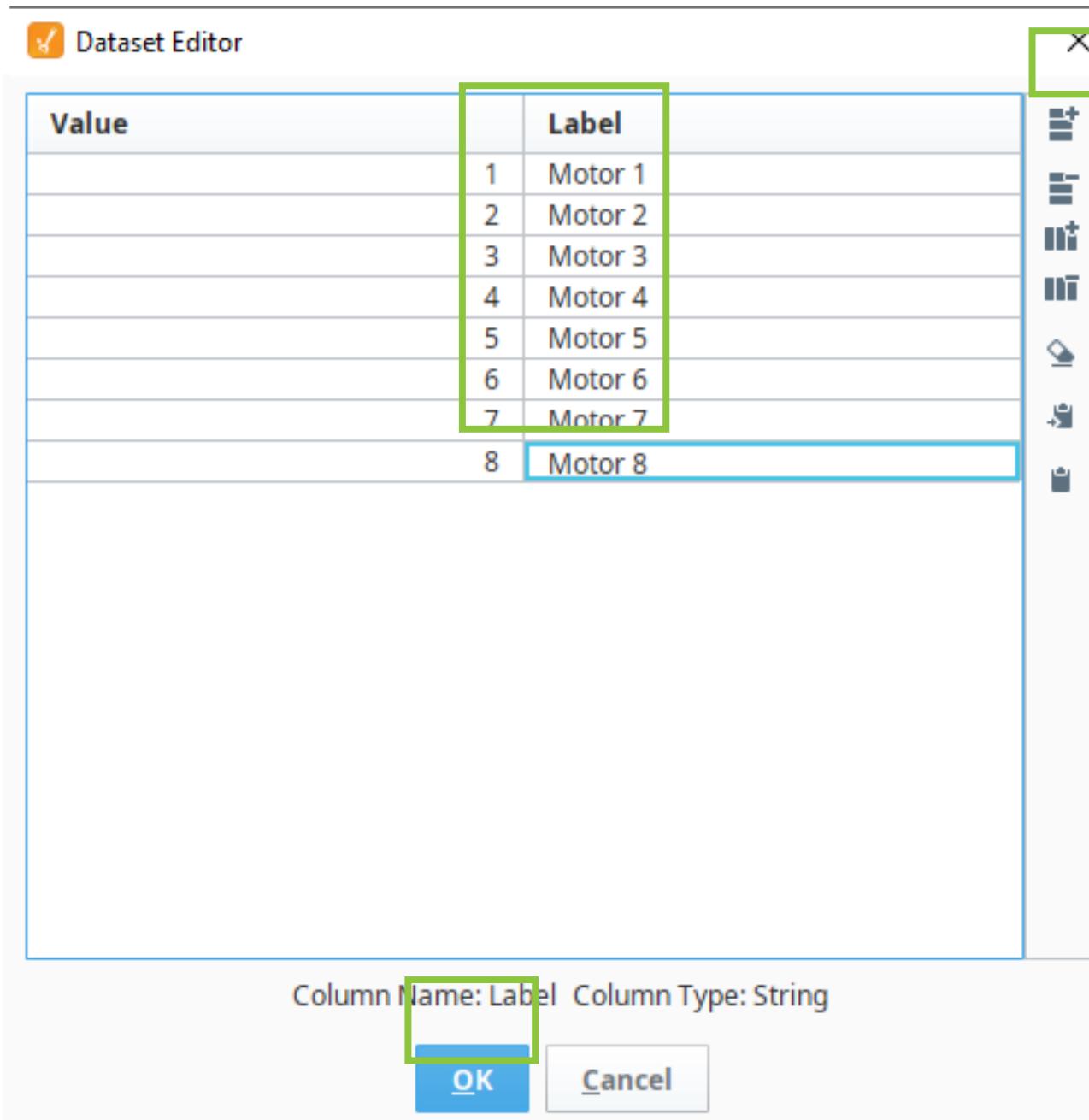
We will add a List component to our Window, and then bind it to the original instance of the Template we put on the Window.

1. Expand the **Components Palette**.
2. Click  to clear the filter.
3. Enter **Dropdown**.
4. Drag a **Dropdown List** component to the Window under the single instance of the Template.
5. Click  on the **Data** property of the List.

Like the Template Repeater, the List is primarily controlled by a dataset. However, unlike the Template Repeater, the List's dataset already has the two columns it needs. The Value column is what is passed out of the List to be used by other components, and the Label column is what the user sees.

6. Click  eight times.
7. Double-click in the first cell of the dataset.
8. Enter **1**.
9. Press **Tab**.
10. Enter **Motor 1**.
11. Press **Tab**.
12. Repeat the steps above to add **Values** and **Labels** for all eight motors.

13. Click **OK**.



Before we move on, we want to test the List.

14. Click ► .

15. Select a **Motor** from the **List**.

Look at the **Selected Value** property on the **Vision Property Editor**. You will see that whenever you select a Motor, this property displays the associated Value from the List. That lets us know what we need to use to bind the Template to the List.

Bind the Template to the List

Now that we have the components we need, we can bind them together.

1. Click 
2. Click the **Template instance**.
3. Click  on the **MotorNum** property.
4. Click **Property**.
5. Click .
6. Expand **Dropdown List**.
7. Click **Selected Value**.
8. Click **OK**.

Now that we have the instance bound to the List, we can test it.

1. Click .
2. Select a **Motor** from the List.

The Template will update to reflect the selected motor.

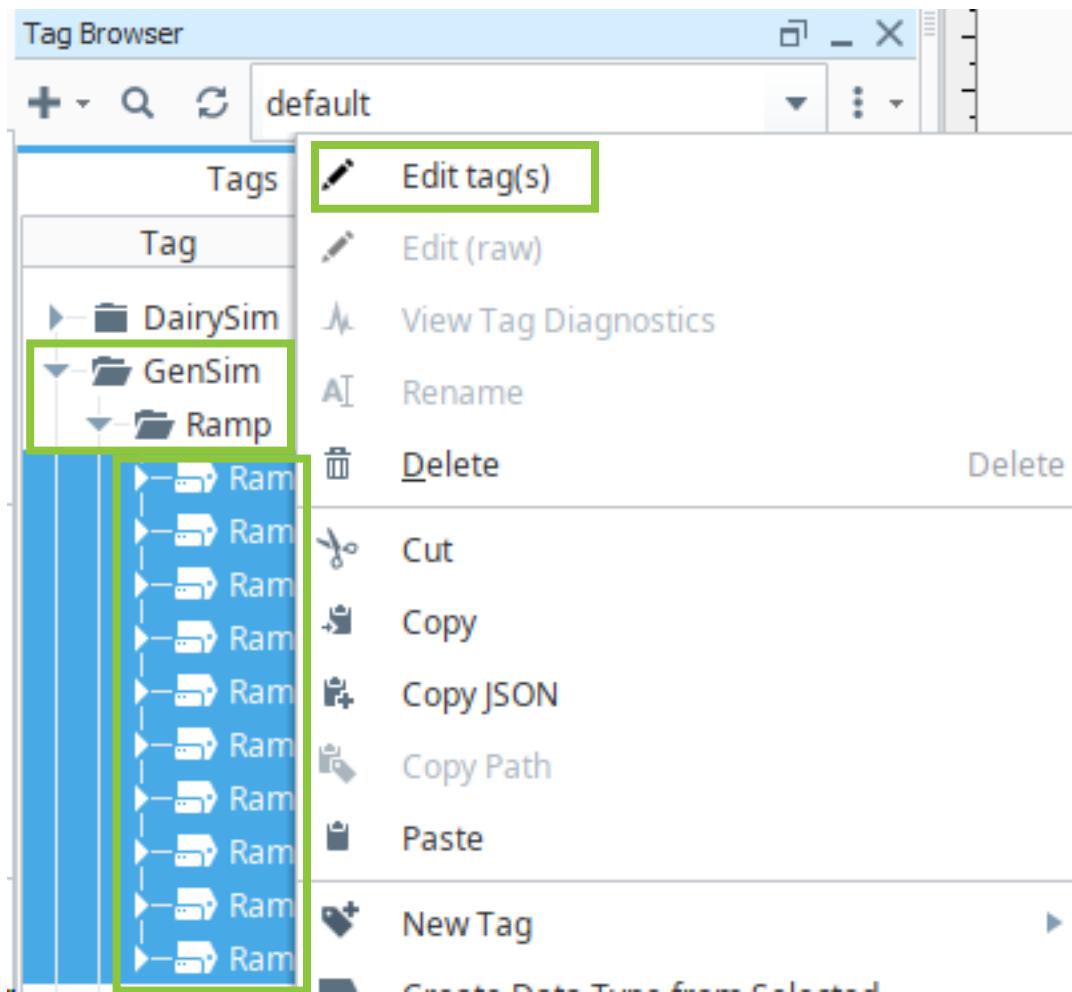
Tag History

Storing History in Ignition is powerful and simple to use. The Tag Historian module uses intelligent algorithms to limit the amount of data being stored while still preserving the ability to perform detailed analytics on the stored data.

Log the Data

Logging data is easy to do with the Tag Historian. Once you enable Tag History, Ignition creates the tables, logs the data, and maintains the database for you.

1. Expand **GenSim** in the Tag Browser.
2. Expand **Ramp**.
3. Click **Ramp0**.
4. Shift-click **Ramp9**.
5. Right-click on any selected Tag.
6. Click **Edit Tag(s)**.



The Tag Editor window opens.

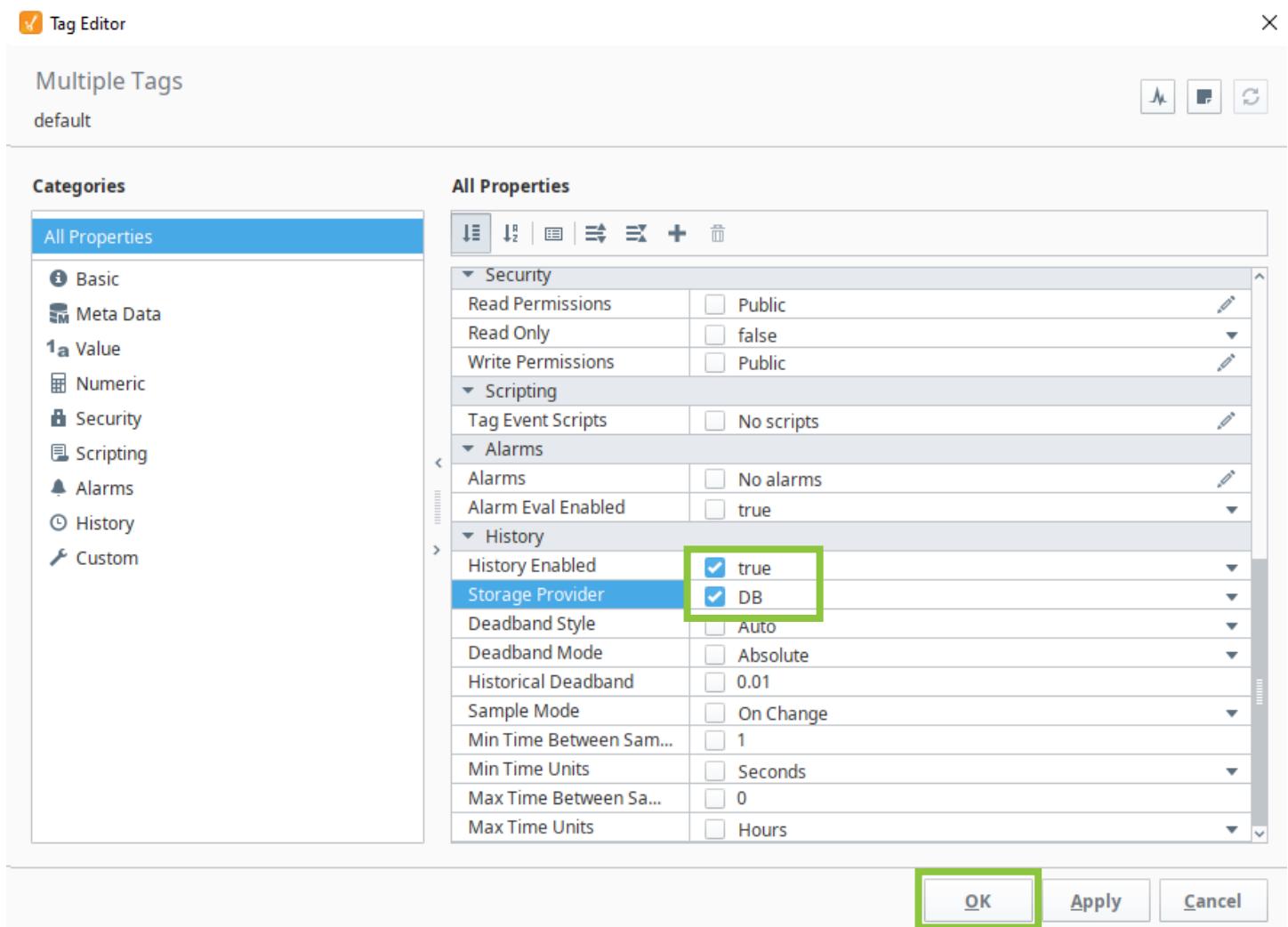
7. Scroll to the bottom of the property list.
8. Change **History Enabled** to **true**.

Note: Do not merely check the checkbox. When you have multiple Tags selected, every property has a checkbox next to it. This merely indicates that the value of that property has changed and will be modified for all selected Tags. In order to enable Tag History, you must change the value of the property from false to true.

9. Select **DB** from the **Storage Provider** list.

Note: You must select a database source as the Storage Provider for Tag History. Because Tag History is a Gateway-level resource, it cannot use the project-level default database connection. If you do not set a data source, Tag History will be enabled but no data will be stored in the database.

10. Click **OK**.



Tag History is now enabled for all of the Ramp Tags. You will now see a clock icon next to each Tag, indicating that History is being stored.

Enable Tag History on a UDT

Enabling Tag History on a UDT automatically enables it for all instances.

1. Click **UDT Definitions** on the Tag Browser.

2. Double-click **MotorUDT**.

The UDT editor opens.

3. Click **Amps**.

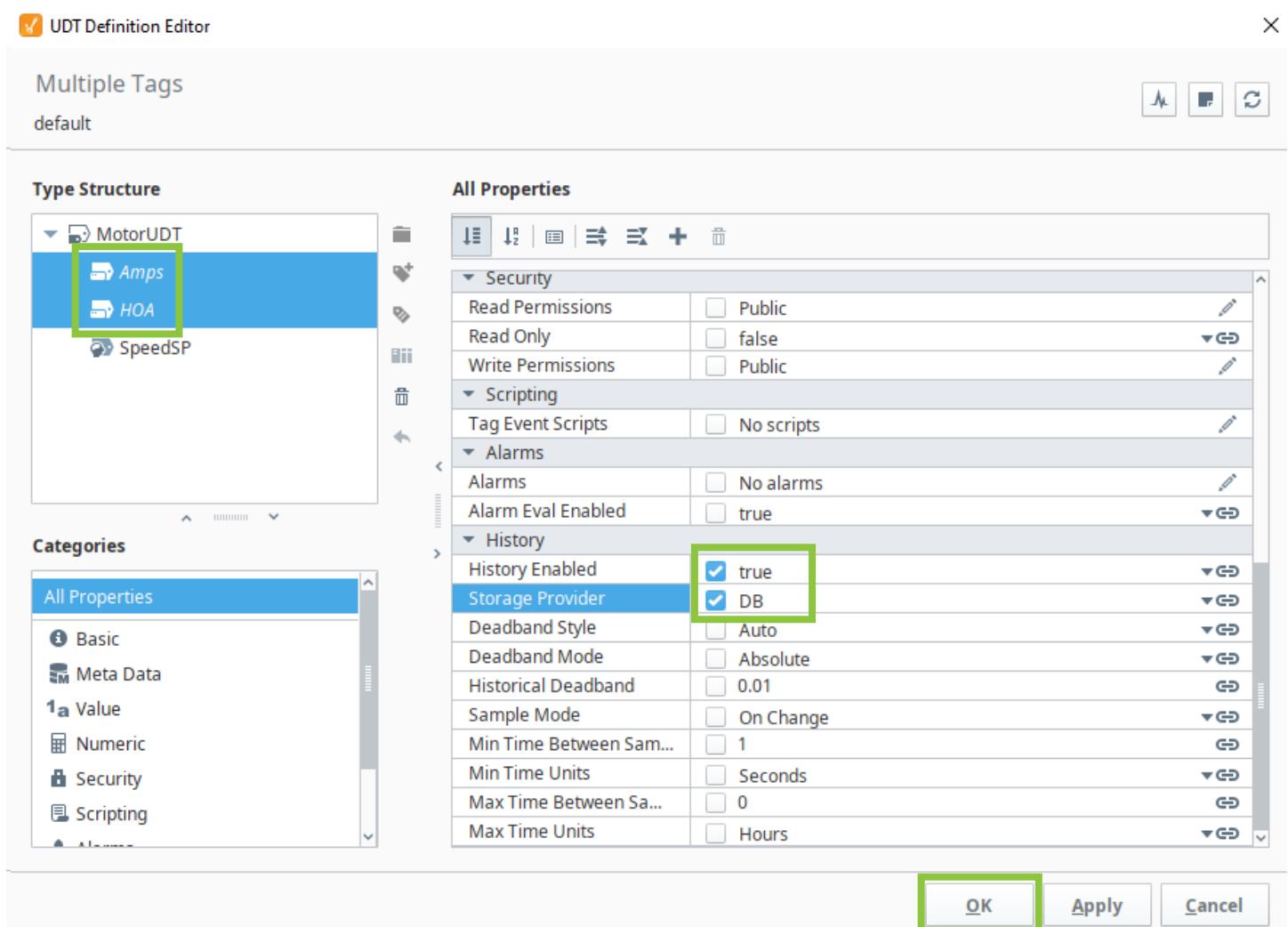
4. Shift-click **HOA**.

5. Scroll to the bottom of the properties.

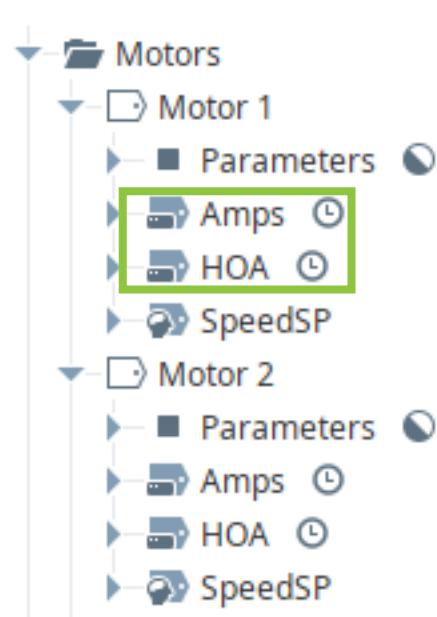
6. Change History Enabled to **true**.

7. Select **DB** from the Storage Provider list.

8. Click **OK**.



9. Click **Tags** on the Tag Browser.
10. Expand any of the Motor instances to confirm that the clock icon is present.



Other Tag History Properties

While the defaults will, more often than not, be sufficient for storing Tag History, it can be helpful to understand the other properties of the Tag Historian.

Deadband The Tag Historian is designed to sample the data, rather than storing all of it. (Transaction groups allow you to store all of the data.) There are two algorithms available for determining how the data is stored: analog and discrete. The default setting for these is determined by the Tag's datatype: Floats and Doubles default to analog, while Booleans, Strings, and Integers default to discrete. The Deadband Style allows you to change this default behavior.

Deadband Mode A setting of Absolute here treats the Historical Deadband setting as an absolute value, while a setting of Percent treats it as a percentage of the Tag's engineering unit span. The Off setting is the equivalent of 0.0.

Historical Deadband This determines the value of the deadband. Only changes that are plus or minus this value will be evaluated for potential storage.

Sample Mode This setting determines how often the Tag Historian should evaluate the value for potential storage. On Change, the default, evaluates every time the value of the Tag changes. Periodic allows you to set a time frame on which the value should be evaluated, regardless of how often the actual value changes. Tag Group allows you to

use a Tag Group to set the frequency on which the Tag's value is evaluated. See "Tag Groups" on page 133 for more information on creating Tag Groups.

Min Time Between Samples Use this setting to set a time that must elapse between samples, regardless of the Sample Mode. For example, if you have Sample Mode set to On Change, and your Tags are changing every quarter second, you could use this setting to have the Historian only evaluate Tags every second.

Min Time Units The unit for the above setting.

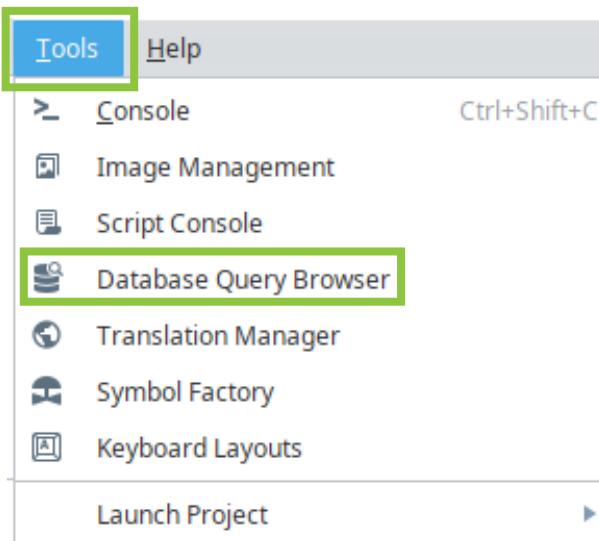
Max Time Between Samples This setting allows you to force the Tag Historian to store a value at some pre-determined rate, even if the other settings would not have triggered storing a value. As an example, imagine a Tag that is storing the temperature in a freezer. If the freezer is working properly, this value will rarely, if ever, change, and thus, the Tag Historian would not store values for the Tag. However, if you were to look at the table, there would not be a way to tell if the values were not being stored because the Tag wasn't changing, or if there was a problem with the Tag Historian. Using this setting, you could force a value to be stored at some rate that allows you to see that the Tag is simply not changing values.

Max Time Units The unit for the above setting.

Viewing the Tag Historian Database Tables

Once enabled, the Tag Historian creates the tables it needs in the specified database. While you should not plan to directly query the database in order to display the data, it can be useful to view the tables to see how the Tag Historian is doing its work.

1. Click **Tools**.
2. Click **Database Query Browser**.



The Database Query Browser opens.

The Database Query Browser is a useful tool within Ignition that allows us to interact with any database that our Gateway has a connection to.

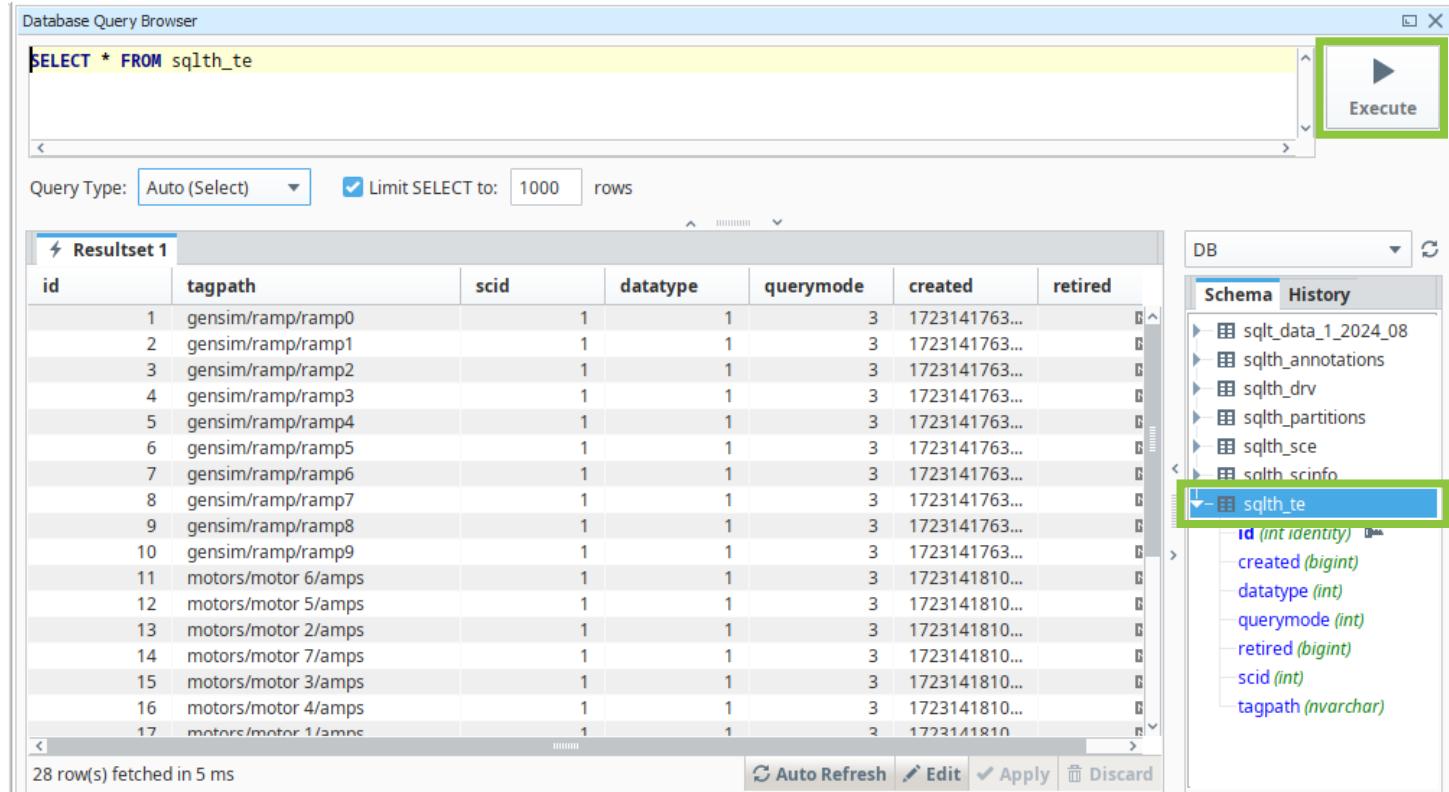
Note: The Database Query Browser is not operating in a “sandbox” environment. Rather, it is a real, live connection to the database. Any changes you make to the data will result in immediate, irrevocable changes to the database.

In the Schema section to the right, you will see that you now have 7 tables, all of them beginning with **sql**.

The 5 tables in the middle of the list—**sqlth_annotations**, **sqlth_drv**, **sqlth_partitions**, **sqlth_sce**, and **sqlth_scinfo**—are tables that the Tag Historian needs for its own internal bookkeeping. None of the information in these tables is designed to be human-readable or useful, and you should not do anything with these tables.

The final table in the list—**sqlth_te**—is useful. Let’s take a look at its data.

1. Double-click **sqlth_te**.
2. Click **Execute**.



The screenshot shows the Database Query Browser interface. In the top-left, the query `SELECT * FROM sqlth_te` is entered. To the right, the **Execute** button is highlighted with a green box. Below the query, the **Query Type** is set to **Auto (Select)** and the **Limit SELECT to:** field is set to **1000 rows**. The main area displays a **Resultset 1** table with 17 rows of data. The columns are **id**, **tagpath**, **scid**, **datatype**, **querymode**, **created**, and **retired**. The data shows various tag paths like `gensim/ramp/ramp0` through `ramp9`, and `motors/motor 1/amps` through `7/amps`. The **DB** pane on the right shows the schema for the **sqlth_te** table, which includes columns **id**, **created**, **datatype**, **querymode**, **retired**, **scid**, and **tagpath**.

id	tagpath	scid	datatype	querymode	created	retired
1	gensim/ramp/ramp0	1	1	3	1723141763...	
2	gensim/ramp/ramp1	1	1	3	1723141763...	
3	gensim/ramp/ramp2	1	1	3	1723141763...	
4	gensim/ramp/ramp3	1	1	3	1723141763...	
5	gensim/ramp/ramp4	1	1	3	1723141763...	
6	gensim/ramp/ramp5	1	1	3	1723141763...	
7	gensim/ramp/ramp6	1	1	3	1723141763...	
8	gensim/ramp/ramp7	1	1	3	1723141763...	
9	gensim/ramp/ramp8	1	1	3	1723141763...	
10	gensim/ramp/ramp9	1	1	3	1723141763...	
11	motors/motor 6/amps	1	1	3	1723141810...	
12	motors/motor 5/amps	1	1	3	1723141810...	
13	motors/motor 2/amps	1	1	3	1723141810...	
14	motors/motor 7/amps	1	1	3	1723141810...	
15	motors/motor 3/amps	1	1	3	1723141810...	
16	motors/motor 4/amps	1	1	3	1723141810...	
17	motors/motor 1/amps	1	1	3	1723141810...	

28 row(s) fetched in 5 ms

When you double-click a table in the Schema section of the Database Query Browser, a SQL statement is automatically written for you that will return all of the data in the table. You can then execute this query without modification to see that data.

What you will see here is a list of the Tags for which you have enabled Tag History; in our case, all 10 Ramp Tags and all of the Tags within each instance of our Motors. There's a lot that isn't important here, but the first two columns are.

id The id of this Tag within the Tag Historian system.

tagpath The Tag being stored.

By matching the id to the Tagpath, we can figure out how to look up the actual data for each Tag.

The remaining table in the query, which should be first alphabetically, is named **sqlt_data_X_XXXX_X**, where the X represents some digit. Careful analysis of those digits will quickly reveal that they are not random. Rather, they represent the current month and year, followed by a counter.

We mentioned previously that the Tag Historian tries to minimize the amount of data being stored in a table in our database. While in theory enterprise-grade databases have only the physical size of the hard drive as a data storage limit, in practice tables with millions or billions of data points can often become unmanageable. One of the strategies the Tag Historian uses to minimize the data is the use of its algorithms (see "Other Tag History Properties" on page 191), but the second, equally important strategy is the use of data partitioning.

By default, the Tag Historian will create a new table to store its data once a month. This is a setting that can be changed in the Gateway, but the default explains the name of this table: the month and year the table is storing data for, and a numeric reference for the portion of that month.

Data partitioning is why you should not plan to directly query the database in order to display the data from the Tag Historian. It simply becomes a nightmare to attempt to know what exact table any given piece of the Tag Historian data resides in any given table. Don't worry, though—there are plenty of components in both Vision and Perspective that are available to you to view this data.

That said, let's go ahead and run a very simple query to see the data that is being stored. Remember that you would not want to do this in a regular environment, but it's a useful way for us to view the limited amount of data we currently have.

1. Find **Ramp0** in the current resultset and note its **id**. It should be **1**.
2. Right-click in the gray area to the right of **Resultset 1**.

3. Click + New Tab.

Resultset 1	
id	tag
1	gensim/ramp/ramp0

4. Double-click the **sqlt_data_X_XXXX_X** table.
5. Enter **WHERE tagid = 1** at the end of the query. If needed, change the number to match the **id** of **Ramp0**.
6. Click **Execute**.

Database Query Browser

```
SELECT * FROM sqlt_data_1_2024_08 WHERE tagid = 1
```

Execute

Query Type: Auto (Select) Limit SELECT to: 1000 rows

Resultset 1						
tagid	intvalue	floatvalue	stringvalue	datevalue	dataintegrity	t_stamp
1	NULL	3.502	NULL	NULL	192	1723
1	NULL	9.905	NULL	NULL	192	1723
1	NULL	0.005	NULL	NULL	192	1723
1	NULL	9.91	NULL	NULL	192	1723
1	NULL	0.01	NULL	NULL	192	1723
1	NULL	9.916	NULL	NULL	192	1723
1	NULL	0.016	NULL	NULL	192	1723
1	NULL	9.922	NULL	NULL	192	1723
1	NULL	0.022	NULL	NULL	192	1723
1	NULL	9.927	NULL	NULL	192	1723
1	NULL	0.027	NULL	NULL	192	1723
1	NULL	9.931	NULL	NULL	192	1723
1	NULL	0.031	NULL	NULL	192	1723

13 row(s) fetched in 9 ms

Auto Refresh Edit Apply Discard

DB Schema History

- sqlt_data_1_2024_08
 - t_stamp (bigint)
 - tagid (int)
 - dataintegrity (int)
 - datevalue (datetime)
 - floatvalue (float)
 - intvalue (bigint)
 - stringvalue (nvarchar)
- sqlt_annotations
- sqlt_drv
- sqlt_partitions
- sqlt_sce
- sqlt_scinfo
- sqlt_te
 - id (int identity)
 - created (bigint)

View the resulting data. Remember that the purpose of the Tag Historian is not to store every value, but rather, to store a sampling of the data. Here, you will see that with the exception of the first record, which is the value that **Ramp0** happened to have been at

the moment you turned on Tag History, the only values being stored at those closest to 10 and closest to 0.

This is as expected, and shows the Tag Historian's analog compression algorithm working as designed.

Display Historian Data

Now that we have seen that we can store Tag History data in our database, let's look at some ways that we can display it.

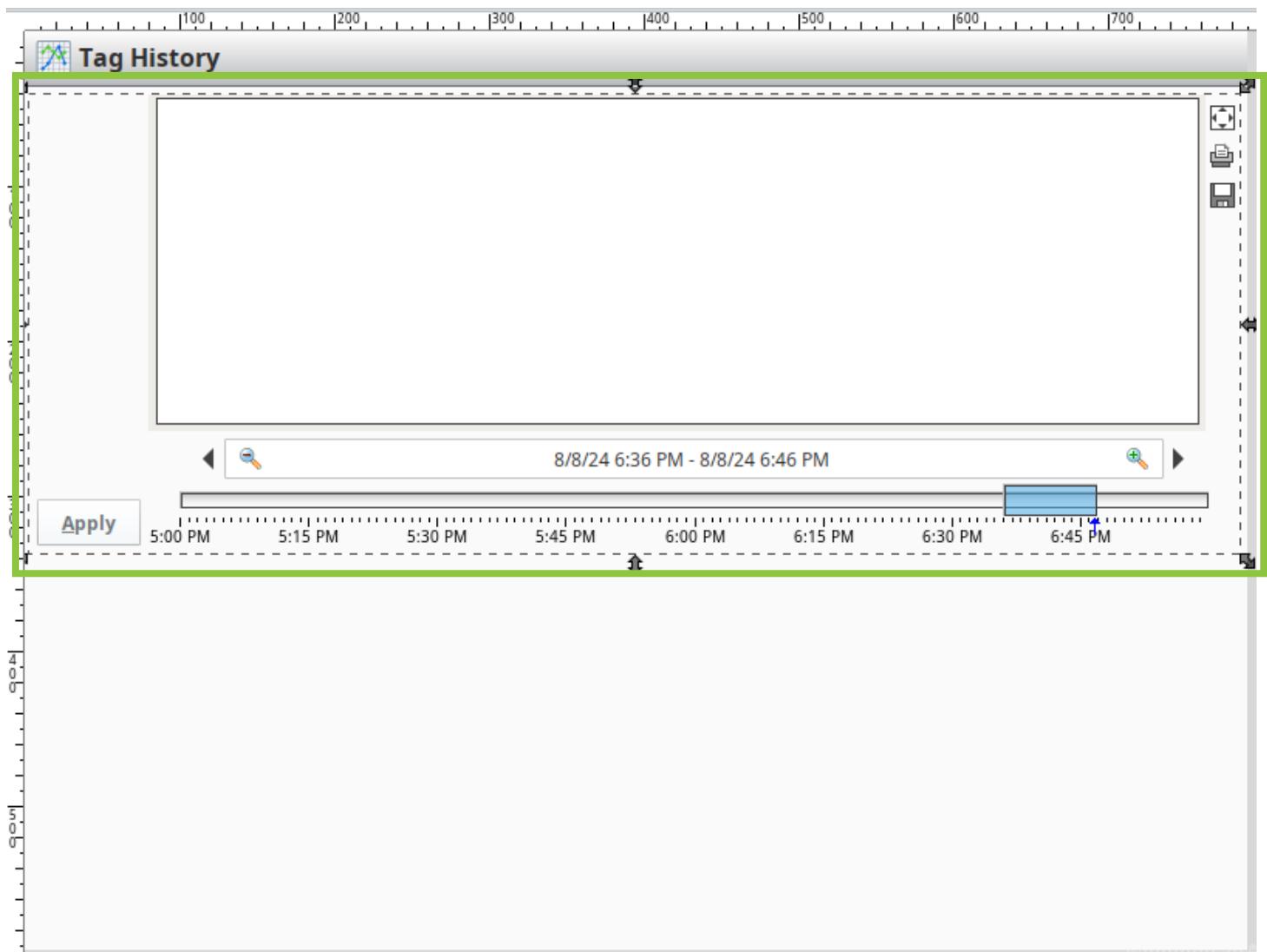
Use an Easy Chart

The first component we will use to display Tag History data is the Easy Chart. Before we create a chart for historical data, we need a new window to display it.

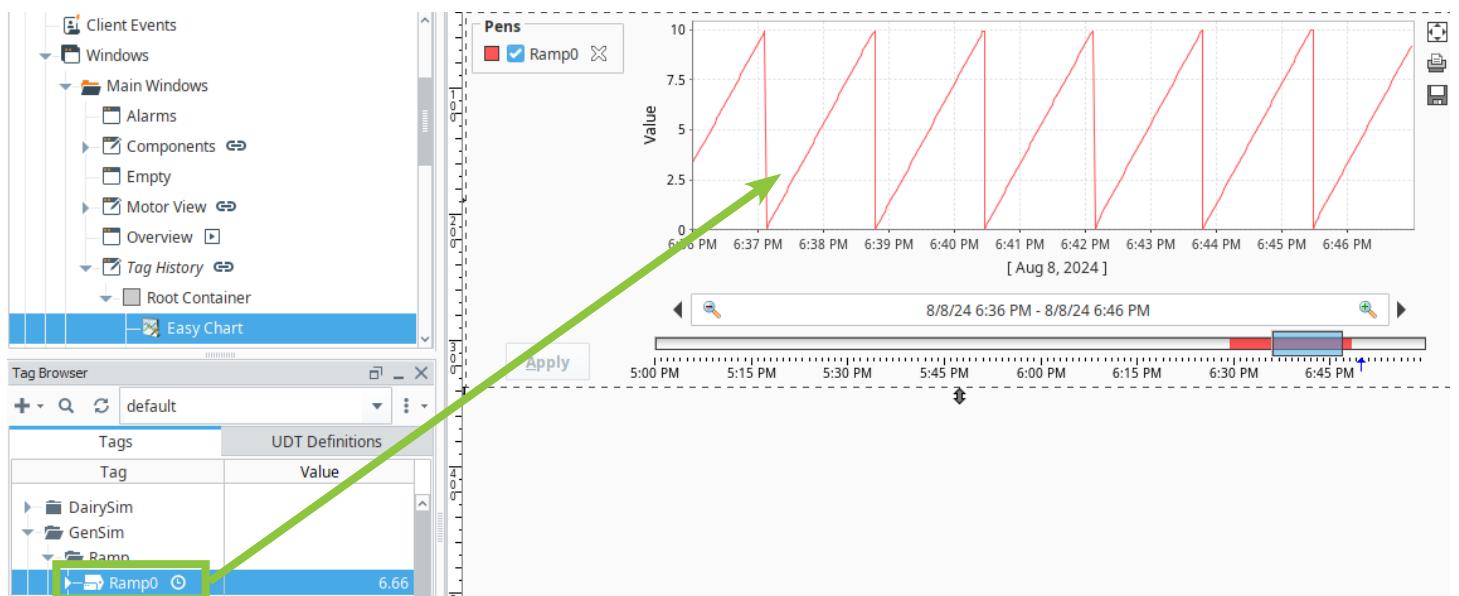
1. Right-click **Empty** in the Project Browser.
2. Click **Duplicate**.
3. Right-click **Empty (1)**.
4. Click **Rename**.
5. Enter **Tag History**.
6. Press **Enter**.
7. Double-click the **header text**.
8. Enter **Tag History**.
9. If desired, change the **header icon**. See "Changing the Header Icon" on page 81.

Now that we have a new window, we can go ahead and add the Easy Chart. This is a component whose primary function is to display Tag History data.

1. Expand the **Components Palette**.
2. Enter **Easy** in the Filter box.
3. Drag an **Easy Chart** to the Window.
4. Resize the chart to fill the **top half** of the window.



5. Drag Ramp0 from the Tag Browser to the Easy Chart.



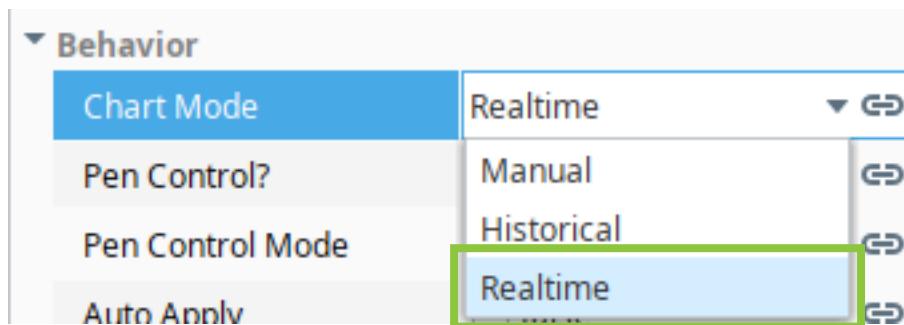
Ch 11. Tag History

Ramp0's Tag History data will display on the chart. Note that this Tag is a simple line graph that grows from 0 to 10, drops back to 0, and repeats.

The Easy Chart has two modes: Historical and Realtime. The Historical mode includes a date range so that operators can select a range of time to view the data. Realtime updates the data continuously, and includes a way to set how far back in time data should be displayed.

The chart defaults to Historical, but let's change that to help us better visualize what is happening.

1. Click the **Easy Chart**.
2. Select **Realtime** from the Mode list.



Now, you will see the chart continuously updating.

This is a useful tool to help us understand the actual data being stored by the Tag Historian. You'll recall that when we looked at the data itself, it was only storing the value closest to 0 and closest to 10. Given that, how is the chart apparently providing new data points every second?

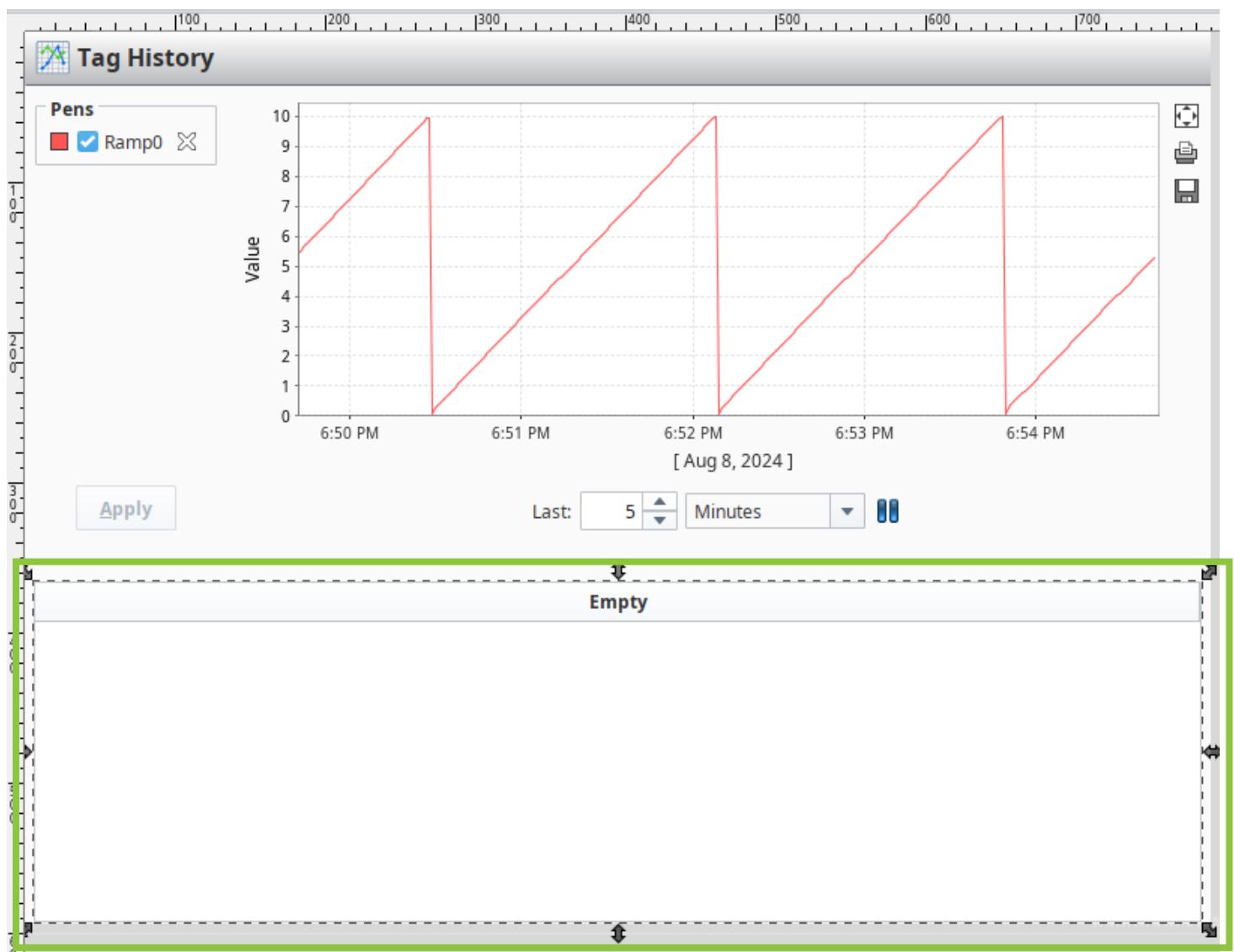
It's critical to understand that the Tag Historian is, by design, *sampling* data. It is *not* designed to store all of the data. Because the Easy Chart is specifically built to display this data, it has the ability to interpolate the data, which is what you are actually seeing on the chart.

Use a Power Table

Another component that can be useful in visualizing Tag History data is the Power Table.

1. Expand the **Components Palette**.
2. Click to clear the filter.
3. Enter **Power**.

4. Drag a **Power Table** to the window.
5. Resize the table to fill the bottom half of the window.



6. Click on the **Data** property.
7. Click **Tag History**.
8. Expand **DB**.
9. Expand the **Gateway**.
10. Expand **gensim**.
11. Expand **ramp**.
12. Click **ramp0**.
13. Shift-click **ramp2**.

14. Drag all three **Tags** to the **Selected Historical Tags**.

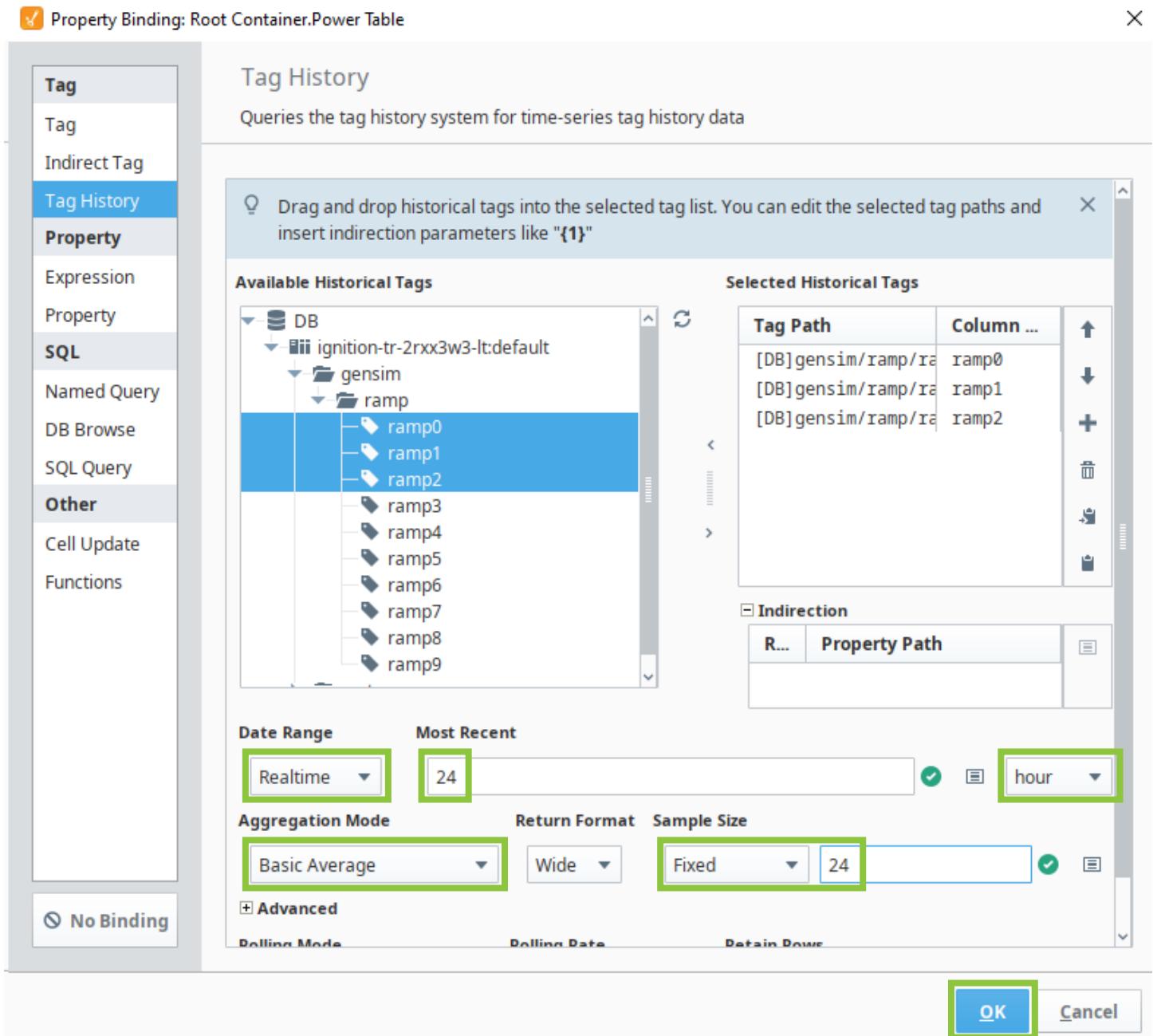
Property Binding: Root Container.Power Table

The screenshot shows the 'Tag History' dialog box. On the left, a sidebar lists options: Tag, Indirect Tag, Tag History (selected), Property, Expression, Property, SQL, Named Query, DB Browse, SQL Query, Other, Cell Update, Functions, and No Binding. The main area is titled 'Tag History' and describes it as 'Queries the tag history system for time-series tag history data'. A tooltip says: 'Drag and drop historical tags into the selected tag list. You can edit the selected tag paths and insert indirection parameters like "{1}"'. Below this are two panes: 'Available Historical Tags' (containing a tree view of DB, ignition-tr-2rxx3w3-lt:default, gensem, ramp, ramp0, ramp1, ramp2, ramp3, ramp4, ramp5, ramp6, ramp7, ramp8, ramp9) and 'Selected Historical Tags' (containing a table with three rows: [DB]gensim/ramp/ramp0, [DB]gensim/ramp/ramp1, [DB]gensim/ramp/ramp2). A green arrow points from the 'ramp0, ramp1, ramp2' items in the available list to the selected list. At the bottom are sections for Date Range (Historical), Aggregation Mode (Min/Max), Advanced settings, and Polling Mode.

The Power Table needs to aggregate the data it will display. By changing the settings on the binding, you can control what this aggregation looks like. We will change the binding to display an average of the last day's worth of data.

15. Select **Realtime** from the **Date Range** dropdown.
16. Enter **24** in the **Most Recent** field.
17. Select **hour** from the dropdown.

18. Select **Basic Average** from the **Aggregation Mode** list.
19. Select **Fixed** from the **Sample Size** list.
20. Enter **24** in the **Sample Size** text field.
21. Click **OK**.



By setting **Most Recent** to 24 and **Sample Size** to a fixed size of 24, we can get one sample per hour for the last 24 hours.

You should now see a table that is mostly filled with zeros. This is expected—we told the table to display the last 24 hours of data, but we have only had Tag History enabled for

the last 1 or possibly 2 hours. To view the data, we need to switch to Preview Mode so we can scroll down.

1. Click ► .
2. Scroll down on the table to see the data.

Because we are averaging the data, the column for Ramp0 should be all 5s, as the average of the data being stored—something close to 0 and something close to 10—will always average out to 5.

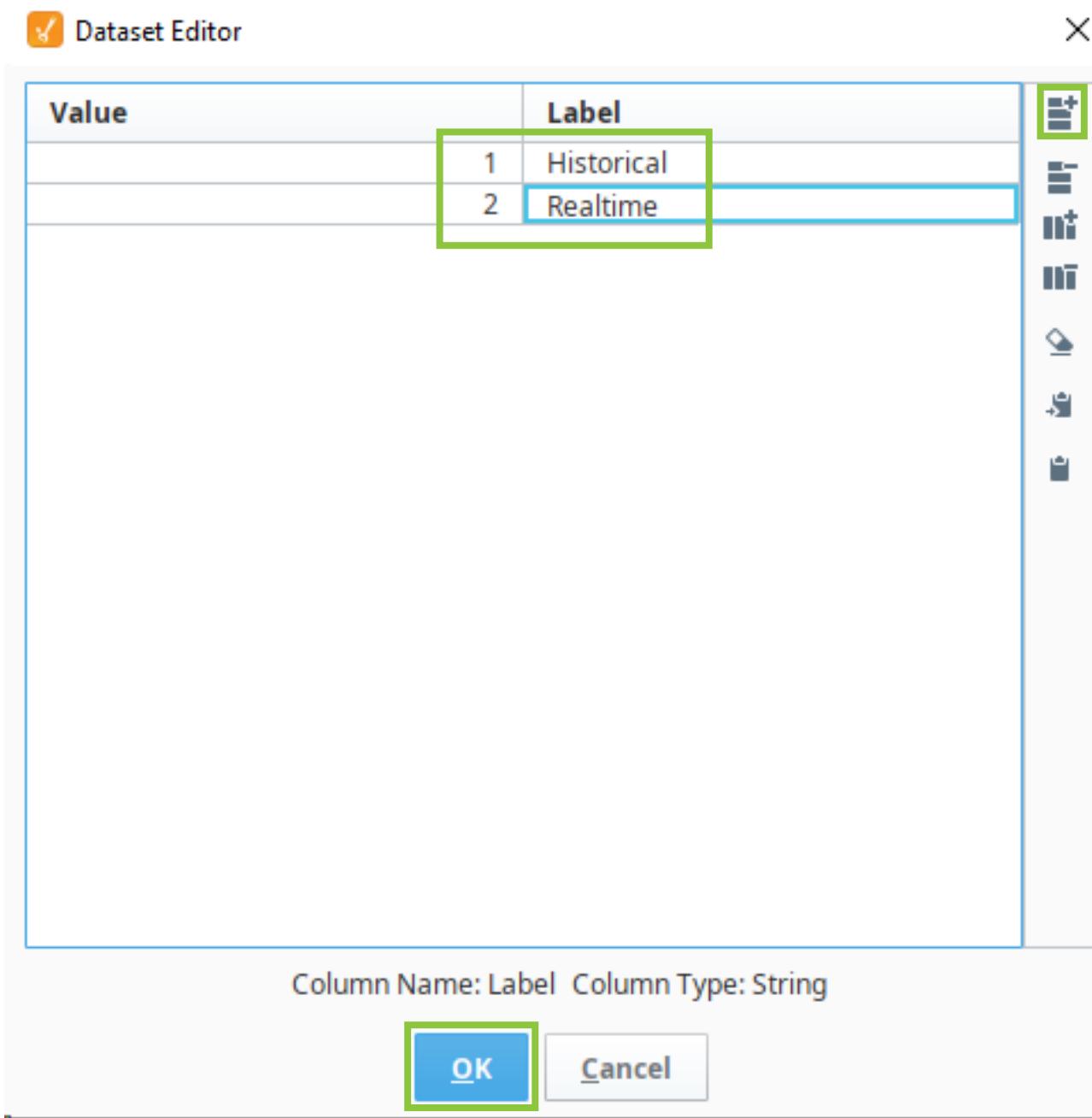
Customize the Easy Chart

We have seen how the Easy Chart makes visualizing Tag History data easy. However, there is a lot more that we can do with this component.

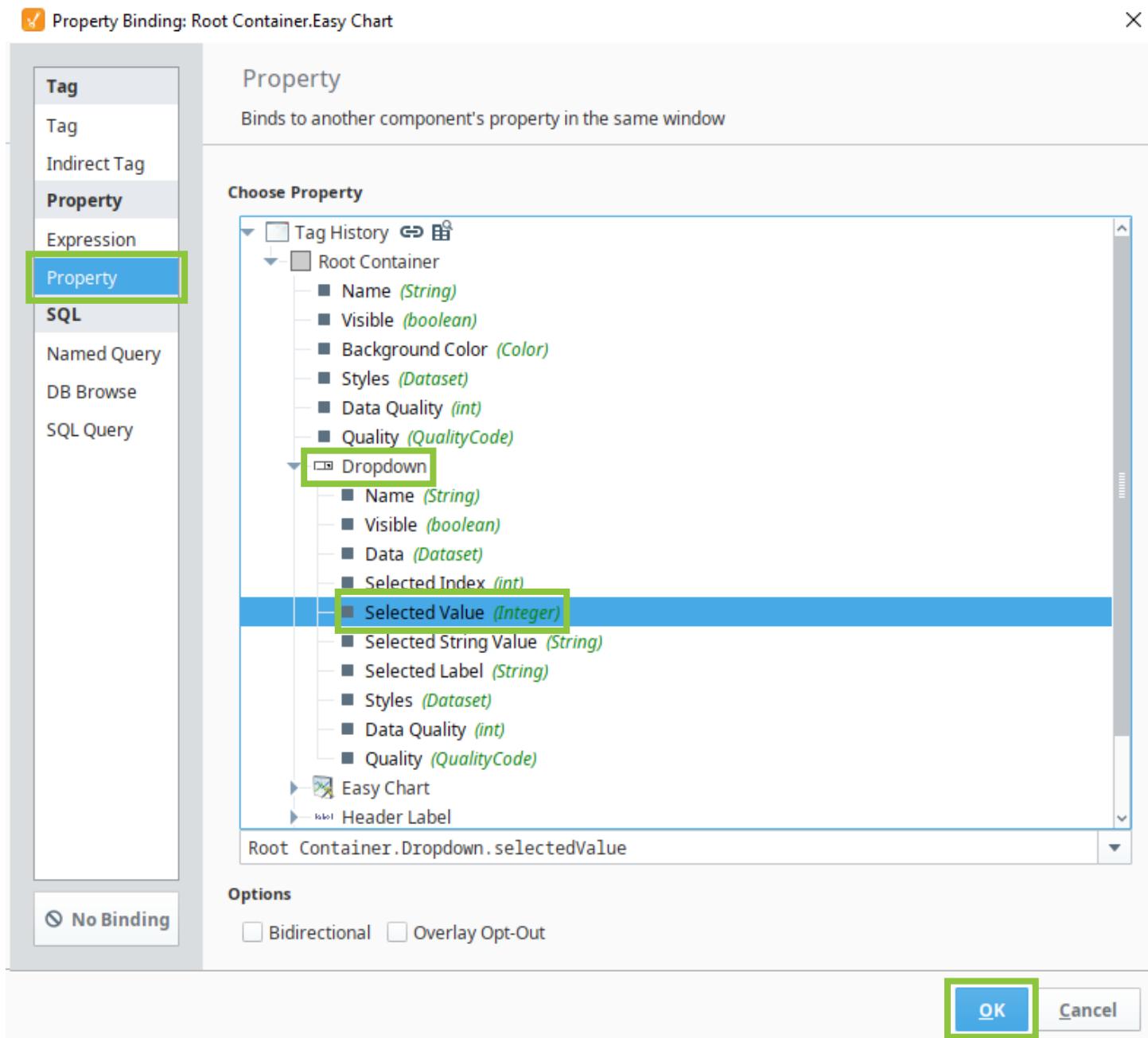
Dynamically Set the Chart Mode

You can allow operators to dynamically set the Chart Mode for the Easy Chart.

1. Resize the **Easy Chart** to make space above it.
2. Expand the **Components Palette**.
3. Click ✖ to clear the filter.
4. Type **dropdown**.
5. Drag a **Dropdown List** component to the window.
6. Click 🔎 for the component's **Data** property.
7. Click 📈 twice.
8. Type **1** in the first Value cell.
9. Press **Tab**.
10. Type **Historical**.
11. Press **Tab**.
12. Type **2**.
13. Press **Tab**.
14. Type **Realtime**.
15. Click **OK**.



16. Click the **Easy Chart**.
17. Click  for the chart's **Mode** property.
18. Click **Property**.
19. Expand **Dropdown**.
20. Click **Selected Value**.
21. Click **OK**.



22. Click ► .

23. Change the selection in the **Dropdown List** to change the Chart's mode.

Allow Operators to Add Tags to the Chart

You can allow operators to add Tags to the chart through a Tag Browse Tree component.

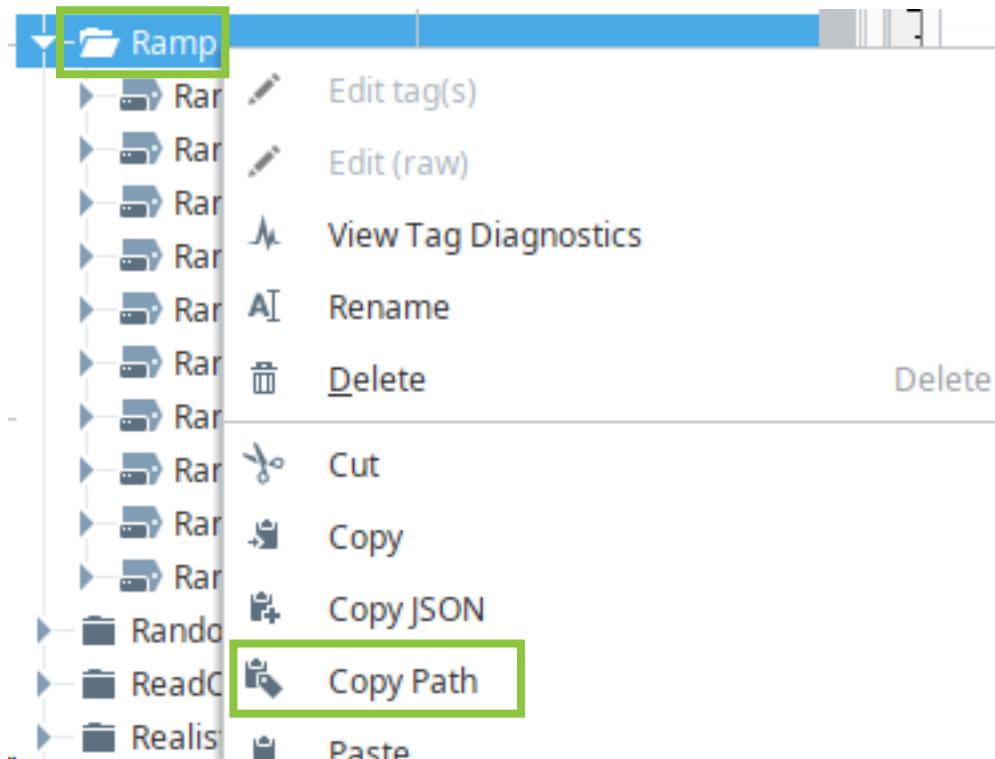
1. Expand the Component Palette.
2. Click ✖ clear the filter.
3. Type **Tag**.

4. Drag a **Tag Browse Tree** component to the window.

Because we are using this in conjunction with the Easy Chart, we want to limit operators to adding Tags that are being historized. But more than that, we want to further limit them to only the Ramp Tags, which we can do by setting the Tree's Root Path property.

5. Right-click the **Ramp** folder in the Tag Browser.

6. Click **Copy Path**.



7. Click in the **Root Node Path** field.
8. Press **ctrl-v** on your keyboard.
9. Uncheck **Include Realtime Tags**.

Data

- Root Node Path: [default]GenSim/Ramp
- Selected Paths: Dataset [1R x 1C]
- Quality

Appearance

- Font: Dialog, Plain, 12
- Tag Tree Mode: Realtime Tag Tree
- Show Root Node: true
- Show Root Handles: true

Realtime Tag Tree Settings

- Include Historical Tags: true
- Include Realtime Tags: false

10. Click ►.
11. Drag one or more of the Tags from the **Tag Tree Component** to the **Easy Chart**.

Restrict Removal of Tags

By default, the Easy Chart allows operators to remove Tags from the chart by clicking ✖ in the Pens legend. You can remove this option so that operators cannot remove pens as easily.

1. Click ■.
2. Click 📄 for the **Tag Pens** property. You will need to scroll to the bottom of the property editor to see this property.
3. Expand the **Dataset Editor** so that it is as wide as your monitor will allow.
4. Click the checkbox on the **USER_REMOVABLE** field for each pen. This is the final column in the dataset.
5. Click **OK**.

Transaction Groups

The other primary method of storing Tag data in a database is Transaction Groups. While Tag History stores a sampling of the data, primarily for trend analysis, Transaction Groups store all of the data. This allows for direct querying of the data tables, and for the retrieval of specific data points, but it can potentially lead to massive amounts of stored data. Transaction Groups can also be bidirectional, allowing for Tags to be changed based on database queries, as in a recipe.

Types of Transaction Groups

There are four types of transaction groups that you can use in your projects.

Historical Group The historical group makes it easy to quickly log historical data to a SQL database. The historical group inserts records of data into a SQL database by mapping items to columns. With full support for triggering, expression items, hour and event meters, and more, you can set up complex historical transactions. Unlike the standard group, the historical group cannot update rows, it can only insert records. Also, it cannot write back to items, except for trigger resets and handshakes.

Standard Group The standard group is a flexible and general use group that can be adapted to a variety of situations. The data model is row-based, with items mapping to columns and the data corresponding to a specific row of a table. The standard group contains items that may be mapped to the database or used internally for features such as triggering or handshakes. Items that are mapped to the database target a specific column of a single specific row, chosen according to the group settings. Items can be mapped either one-way or bidirectionally. The group may also insert new rows instead of updating a specific row. In this way, data can be inserted for historical purposes based on a timer, with an optional trigger.

Block Data Group The block data group is so named because it writes “blocks” of data to a table, consisting of multiple rows and columns. A block data group contains one or more block items. Each block item maps to a column in the group’s table. Then it defines any number of values (OPC or Tag items) that will be written vertically as rows under that column. Block groups are very efficient and they can be used to store massive amounts of data to the database. For example, 100 columns each with 100 rows would give us 100,000 data points, which will often only take a few hundred milliseconds to write, depending on the database. Like the standard group, the block group can insert a new block or update the first, last, or a custom block. Additionally, the group can be set to only insert rows that have changed in the block.

Stored Procedure Group The stored procedure group is similar to the other groups in terms of execution, triggering, and item configuration. The primary difference is that unlike the other group types, the target is not a database table, but instead a stored procedure. Items in the group can be mapped to input or output parameters of the procedure. They can also be bound to output parameters, in which case the value returned from the procedure will be written to the item. Items can be bound to both an input and output at the same time.

Group Items

Items are the core elements of a group. When items are executed, the values are used by the group (for logic purposes), by other items, and to write to the database.

Items can be written to and from the database or from other items. The following are the group items types:

OPC Item An OPC item, directly subscribes to an OPC server at the rate of the group. Executed by the group, so alarms are evaluated when the group is executed. These items are executed even when the trigger isn't active.

Expression Item (Run-Always) Run-Always expression items are much like an expression Tag. They can be either a static value, an expression, or a database query. Run-Always expression items are evaluated at each group interval before the trigger state is evaluated.

Expression Item (On-Trigger) On-Trigger expression items are the same as Run-Always expression items, except that they are only executed after the trigger has been evaluated and is active.

Tag Reference Allows a Tag to be used in a group like any other item type, except that the Tag is evaluated by its Tag Group instead of by the group.

It is easy to confuse the definition and purpose of Tags and OPC items in transaction groups, though they have distinct benefits.

Tags may be referenced inside of groups, however it is critical to remember that they are executed by the Ignition Gateway, according to their scan classes and independent of the group. Adding a Tag into a group is like creating a shortcut to that Tag.

Triggers

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

The trigger settings are the same for all group types. They are found in the right side of the Transaction Group workspace, on the second tab.

There are two types of triggers. One type only evaluates when values have changed, while the other executes the group based on the value of an item within the group.

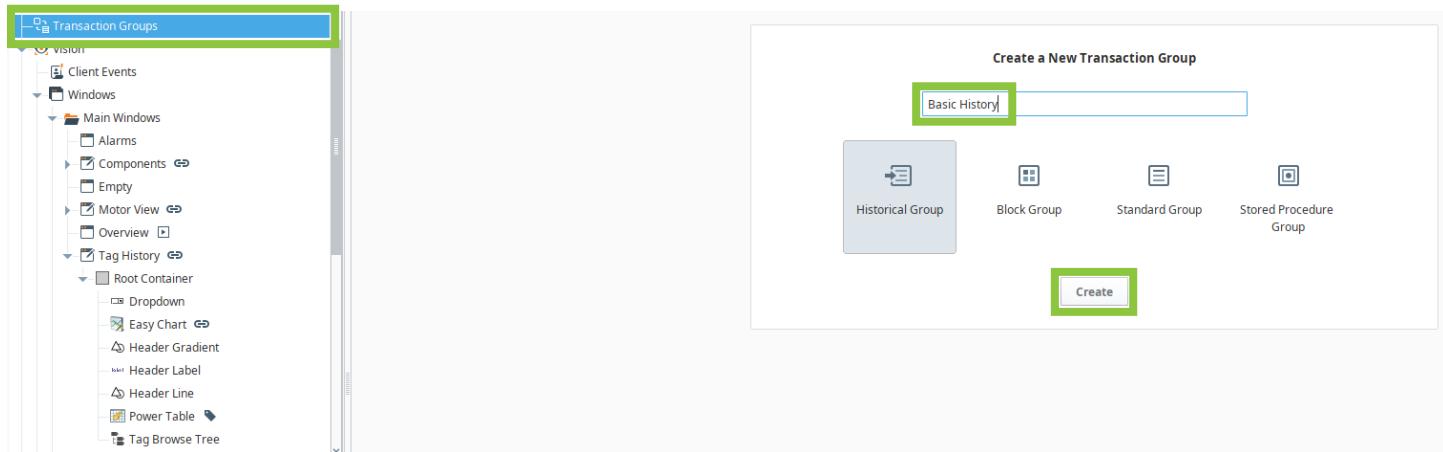
Creating a Basic History Group

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

Create an Historical Group

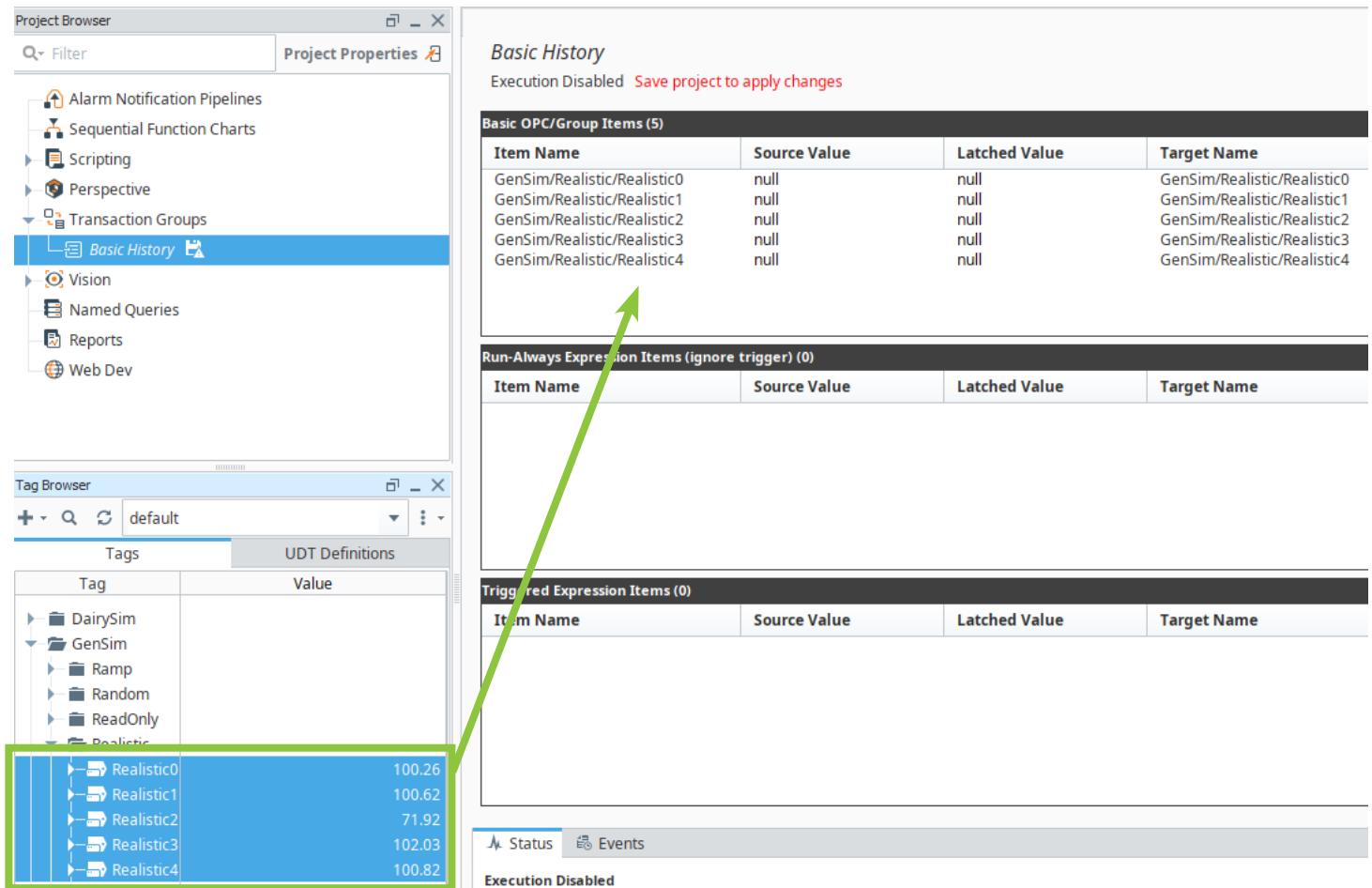
Let's create an historical group that logs a few Tags to the database every 10 seconds.

1. Click **Transaction Groups** in the Project Browser.
2. Enter **Basic History** in the Name of the transaction group field.
3. Click **Create**.



Now we need to drag in the Tags we want to log.

4. Expand **GenSim** in the Tag Browser.
5. Expand **Realistic**.
6. Click **Realistic0**.
7. Shift-click **Realistic4**.
8. Drag the selected Tags to the **Basic OPC/Group Items** table.



Configure the Action Settings

Next, we need to setup the group rate and database connection.

1. Enter **10** in the **Timer** text field.
2. Leave the **Data source** list at **<Default>**.

Note: Because Transaction Groups are part of the project, they can use the project default database connection.

3. Enter **history** for the table name.
4. Press **Enter**.

Note: If you do not press Enter after typing the name for the table name, it may not set the name and instead will revert back to the default group_table.

The screenshot shows the configuration interface for a Transaction Group. At the top, there are three tabs: Action (selected), Trigger, and Options. Under Execution Scheduling, the Timer option is selected, with a value of 10 and a unit of second(s). The Data source is set to <Default>. The Table name is history. There are several configuration options below:

- Automatically create table
- Use custom index column: history_ndx
- Store timestamp to: t_stamp
- Store quality code to: quality_code
- Delete records older than: 1 day(s)

Run the Group

Now that we have the basic settings configured, we can run the group to make sure it works as we expect it to.

1. Click **Enabled**.
2. Click **File**.
3. Click **Save All**.

The group will be enabled and will start to log data. You should see the values on Total Executions and DB Writes start to increment every 10 seconds. You will also see a green arrow next to the group on the Project Browser.

Examine the Data

Let's take a look at the data being stored by the group.

1. Click **Disabled**.

2. Click **File**.

3. Click **Save All**.

The group stops executing.

4. Click **Tools**.

5. Click **Database Query Browser**.

The Database Query Browser opens.

6. Double-click **history**.

7. Click **Execute**.

The screenshot shows the Database Query Browser interface. At the top, there is a toolbar with a 'Database Query Browser' title bar. Below the toolbar, a query window contains the SQL command: `SELECT * FROM history`. To the right of the query window is an 'Execute' button. Underneath the query window, there are two tabs: 'Resultset 1' and 'DB'. The 'Resultset 1' tab displays a table with the following data:

history_ndx	Realistic0	Realistic1	Realistic2	Realistic3	Realistic4	t_stamp
1	101.361	99.856	95.171	98.699	100.835	2024-08-08 21:12:58....
2	101.85	99.721	97.677	99.671	100.276	2024-08-08 21:13:08....
3	104.206	100.896	146.169	99.106	98.878	2024-08-08 21:13:18....
4	98.349	101.813	93.452	102.372	98.806	2024-08-08 21:13:28....
5	99.163	99.337	88.046	99.375	99.91	2024-08-08 21:13:38....

Below the table, a message says '5 row(s) fetched in 4 ms'. To the right of the table is a 'Schema' pane showing the structure of the 'history' table and other database objects like 'sqlth_annotations' and 'sqlth_partitions'.

The table shows all of the data that was logged by the group.

It's important to note that unlike Tag History, the group is logging all of the data: each individual data point is recorded for each Tag.

This might be what is needed by your organization, but it's also important to note that all of this data is being logging a single table in the database. Transaction Groups can very quickly result in unmanageably large tables.

That said, the fact that the group is being logged in a single table—one that we named—does mean that, unlike the Tag Historian's data, we can write SQL statements directly against this data.

Add a Trigger

A Transaction Group can be dynamically controlled through the use of a trigger. With a trigger, we can only have the Transaction Group become active when one or more of its Tags are actually changing values, which will prevent it from needlessly recording duplicate values when the Tags are not changing, or we can turn the Group on and off by changing the value of one of the Group's Tags, which is what we will explore in our example.

1. Ensure your Designer is in **Read-Write mode**.



2. Click **TempF** in the Tag Browser.
3. Double-click its **value**.
4. Enter **50**.
5. Drag **TempF** to the **Basic OPC/Group Items** table.

The screenshot shows the Tag Browser interface. On the left, there's a navigation tree with 'Basic History' selected, containing 'Vision', 'Named Queries', 'Reports', and 'Web Dev'. Below it is the 'Tag Browser' window with 'default' selected. The 'Tags' tab is active, showing a table with columns 'Tag' and 'Value'. The table contains entries for 'DairySim', 'GenSim', 'Motors', 'TempC', and 'TempF'. The 'TempF' row is highlighted with a blue background. A green arrow points from this row to the 'TempF' entry in the 'Run-Always Expression Items' table on the right.

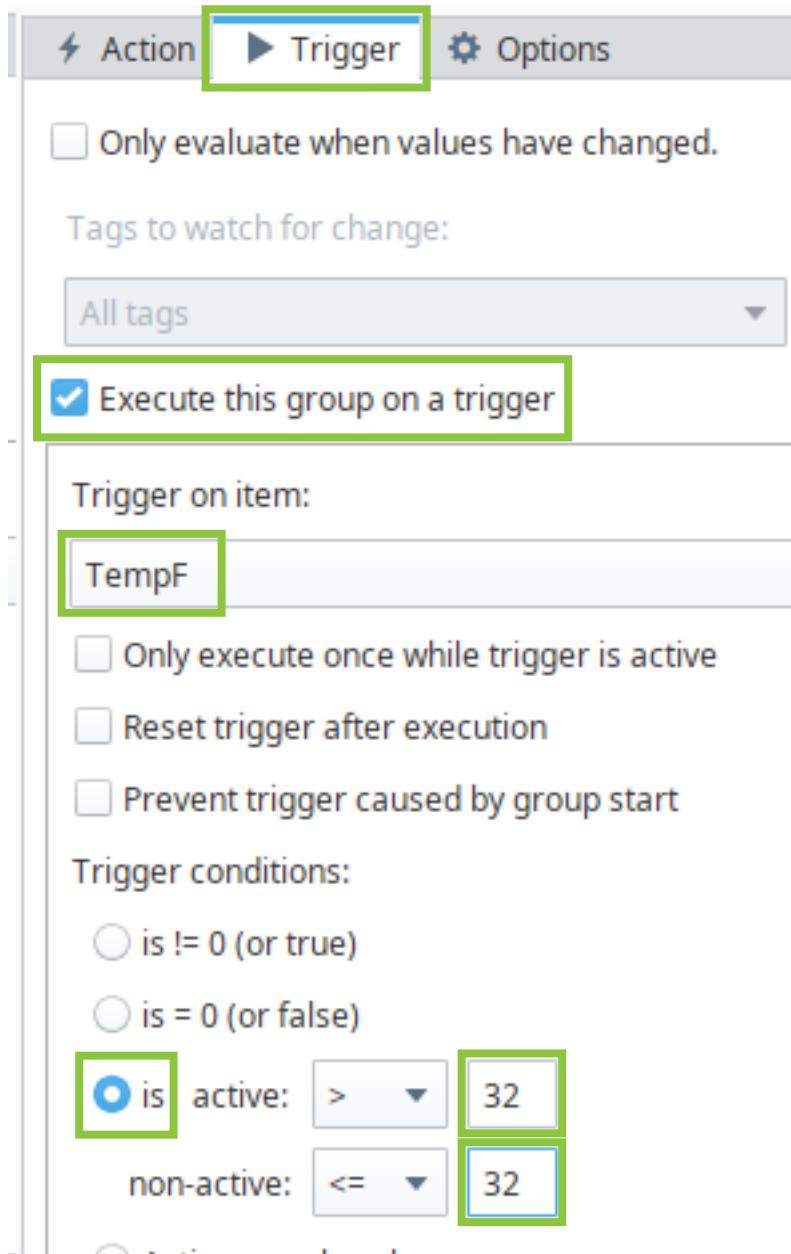
Item Name	Source Value	Latched
GenSim/Realistic/Realistic3	null	null
GenSim/Realistic/Realistic4	null	null
TempF	null	null

Item Name	Source Value	Latched

Item Name	Source Value	Latched

6. Click **Trigger**.
7. Click **Execute this group on a trigger**.
8. Select **TempF** from the **Trigger on item** list.
9. Select **is active** from the **Trigger conditions**.
10. Enter **32** in the **>** field.
11. Enter **32** in the **<=** field.

Note: Be careful that your conditions do not overlap or leave a gap in values.



12. Click **Enabled**.

13. Click **File**.

14. Click **Save All**.

The Transaction Group will resume, and you will once again see the Total executions and DB writes values incrementing every 10 seconds.

Now, let's use the trigger to dynamically stop the group.

1. Double-click the value of **TempF** in the Tag Browser.
2. Enter **20**.

3. Press **Enter**.

Because the trigger's value is now below the value set to make it non-active, it will stop executing. It is still Enabled, though—you'll still see the green arrow on the Project Browser—so that at any point when the trigger once again raised above the threshold, it'll begin recording values again.

We will use this data later in the class, so we want to make sure that the Transaction Group is recording values, so let's get it going again by resetting our trigger.

1. Double-click the value of **TempF** in the Tag Browser.
2. Enter **50**.
3. Press **Enter**.

Creating a Recipe Group

While recording historical data is a very common use of Transaction Groups, another common use is to leverage the ability of a group to be bidirectional: not only can groups write Tag information to the database, they can also read information from the database, and then use that information to write values back to Tags. A very common use-case for this feature is loading recipes.

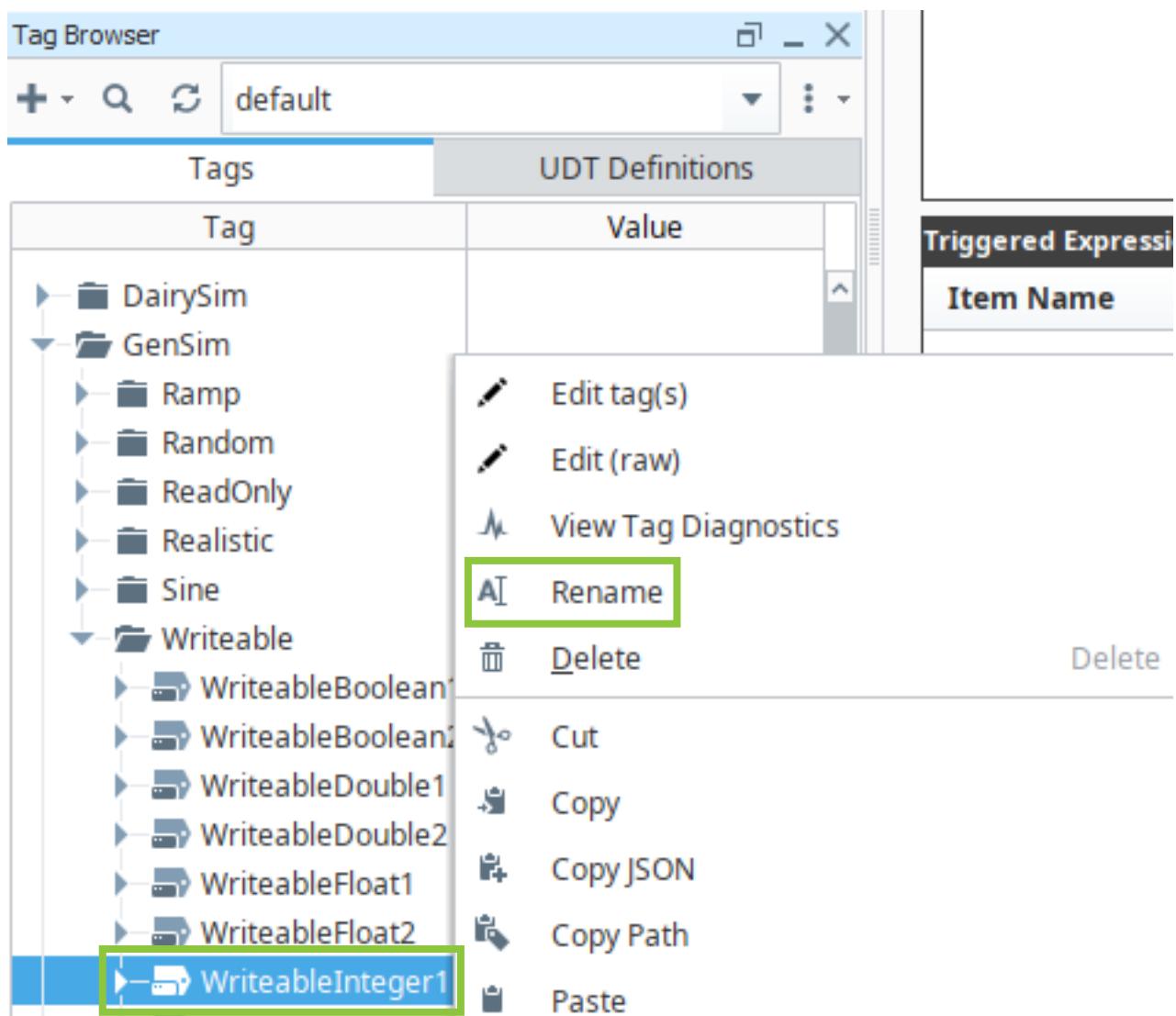
To set this up, we're going to have the Transaction Group create a new table in our database with column names derived from some Tags. Then, we'll manually populate the database with some (made up) recipe information. After that, we'll run the group and have it write values back to the Tags based on the information in the database. Don't worry, it's not as complex as it sounds.

Rename the Tags

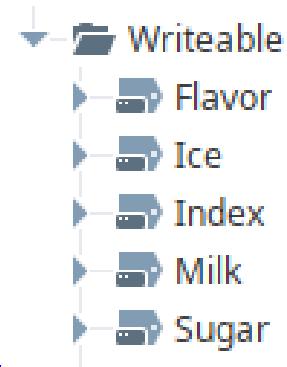
To make this example a little more fun, let's create a recipe for several flavors of ice cream. Ice cream needs four basic things: milk, sugar, ice, and a flavor.

We'll start by renaming some of the Tags in the Writable folder to stand in for our ingredients. We'll use three of the WriteableInteger Tags for milk, sugar, and ice, and one of the WriteableStrings for the flavor. We also need a fourth WriteableInteger to represent the index, or primary key, of the recipe in the database table we'll eventually use.

1. Expand **GenSim** in the Tag Browser.
2. Expand **Writable**.
3. Right-click **WriteableInteger1**.

4. Click **Rename**.5. Enter **Milk**.6. Press **Enter**.7. Right-click **WriteableInteger2**.8. Click **Rename**.9. Enter **Sugar**.10. Press **Enter**.11. Right-click **WriteableInteger3**.12. Click **Rename**.13. Enter **Ice**.

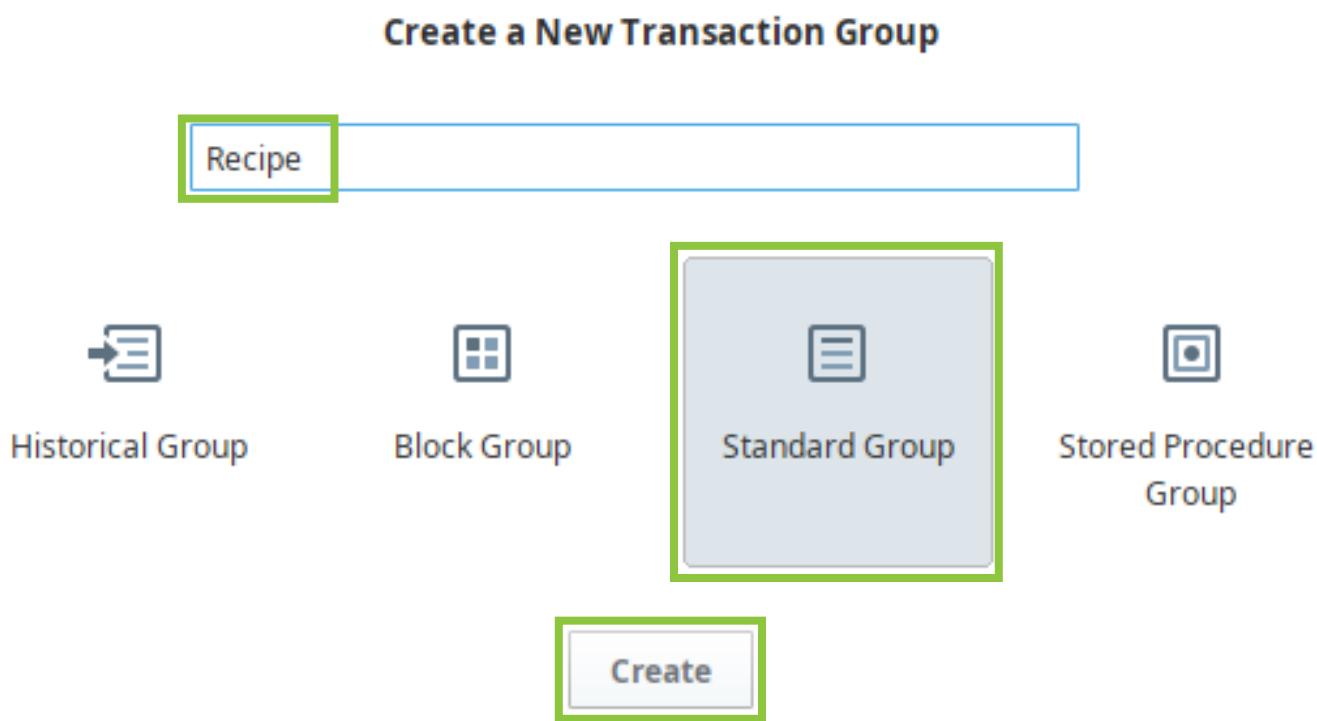
14. Press **Enter**.
15. Right-click **WriteableInteger4**.
16. Click **Rename**.
17. Enter **Index**.
18. Press **Enter**.
19. Right-click **WriteableString1**.
20. Click **Rename**.
21. Enter **Flavor**.
22. Press **Enter**.



Create the Transaction Group

Now that we have our Tags in place, we need to create the group itself. Because we need bidirectionality, we will use a Standard Group.

1. Click **Transaction Groups** in the Project Browser.
2. Enter **Recipe** in the **Name of the transaction group** field.
3. Click **Standard Group**.
4. Click **Create**.



5. Click **Flavor**.
6. Shift-click **Sugar**.
7. Drag all five Tags to the **Basic OPC/Group Items** table.

The screenshot shows the Tag Browser interface with a tree view of tags and a table view of specific tags. A green arrow points from the 'Writeable' folder in the Tag Browser to the 'Basic OPC/Group Items' table in the main window.

Tag Browser:

- Sequential Function Charts
- Scripting
- Perspective
- Transaction Groups
 - Basic History
- Recipe
- Vision
- Named Queries
- Reports
- Web Dev

Table View (Tags):

Tag	Value
DairySim	
GenSim	
Ramp	
Random	
ReadOnly	
Realistic	
Sine	
Writeable	
Flavor	0
Ice	0
Index	0
Milk	0
Sugar	0

Main Window (OPC/Group Items):

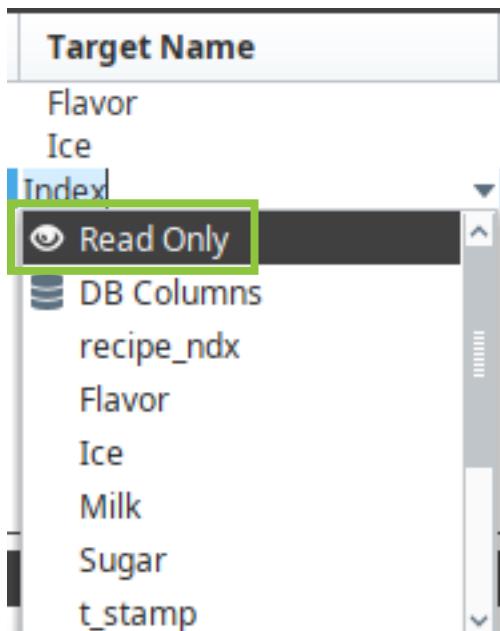
Item Name	Source Value
GenSim/Writeable/Flavor	0
GenSim/Writeable/Ice	0
GenSim/Writeable/Index	0
GenSim/Writeable/Milk	0
GenSim/Writeable/Sugar	0

8. Enter **recipe** in the Table name field.
9. Press **Enter**.
10. Click **Store timestamp to** so that it is not selected.

Set Target Name

Each column in the Basic OPC/Group Items table has a Target Name setting. This will be the name that is assigned in the database table when it gets created. In our case, we want the defaults—the name of the Tag—for each column, except the Index. When the Transaction Group creates the table, an index (primary key) column will be created. So, we do not want an additional column for our Index Tag. We'll associate that with the primary key field in a later step. For now, we need to tell the Transaction Group to not create a field for that Tag.

1. Select **Read Only** from the Target Name list.



Create the Table

While it's possible to create the table needed for the recipe directly in the database, we can save ourselves a lot of work by letting the Transaction Group create the table. Then, we can go into the table and delete the data, then replace it with our own data.

1. Click **Enabled**.
2. Click **File**.
3. Click **Save All**.

Allow the group to run for several seconds, enough for at least 4 total executions.

4. Click **Disabled**.
5. Click **File**.

6. Click **Save All**.

Now that the table has been created, we can use the Database Query Browser to update the data to the values we need.

7. Click **Tools**.

8. Click **Database Query Browser**.

9. Double-click **recipes**.

10. Click **Execute**.

The screenshot shows the Database Query Browser interface. In the top-left, there's a code editor window containing the SQL query: `SELECT * FROM recipe`. Below it, the query type is set to "Auto (Select)" and a limit of "1000 rows" is specified. To the right of the code editor is a large button labeled "Execute".

In the center, the "Resultset 1" table is displayed with the following data:

recipe_ndx	Flavor	Ice	Milk	Sugar
1		0	0	0
2		0	0	0
3		0	0	0
4		0	0	0
5		0	0	0
6		0	0	0
7		0	0	0

On the right side of the interface, there's a "Schema" tree view. It shows the database structure with the "recipe" table selected. The "recipe" table is defined with four columns: "recipe_ndx" (int identity), "Flavor" (nvarchar), "Ice" (int), "Milk" (int), and "Sugar" (int). Below the schema tree, there are other tables like "sqlt_data_1_2024_08" and various log and system tables.

You can see that by letting the Transaction Group run, it created the table and the rows we need. It also populated the table with a bunch of data. Note here that even though the Tags are not changing—none of the Writeable Tags change values automatically—the group was still logging those values every second. You'll also see that a column was not created for the Index Tag.

Update the Data

Our next step is to delete that data and any extra rows we don't need and replace it with our own data. Thankfully, this can all be done in the Database Query Browser without writing code.

1. Click **Edit**.

2. Click in the **first cell** in row **5**.

3. Shift-click the **first cell** in the **last row** of the table.

Note: the number of rows you have will vary depending on how long you let the Transaction Group run. We want to delete all but 4 rows, so the number of rows you are deleting does not matter.

4. Right-click in **one of the selected rows**.
5. Click **Delete Row(s)**.
6. Click **Apply**.

The screenshot shows a Database Query Browser window. The query in the top pane is:

```
SELECT * FROM recipe
```

The results pane shows a table named 'Resultset 1' with columns: recipe_ndx, Flavor, Ice, Milk, Sugar. The data is as follows:

recipe_ndx	Flavor	Ice	Milk	Sugar
1		0	0	0
2		0	0	0
3		0	0	0
4		0	0	0
5		0	0	0
6		0	0	0

A context menu is open over the last row (row 6), with the 'Delete Row(s)' option highlighted by a green box.

The right pane shows the database schema with the 'recipe' table selected. The 'Edit' button at the bottom of the results pane is also highlighted with a green box.

Now, we can change the data to represent values we actually need. Feel free to make up all of the values, including the flavors. Just make sure that each numeric value is different, so that when we run the group, you will be able to see that the values are changing.

7. Double-click in the **first cell** in the **Flavor column**.
8. Enter **Vanilla**.
9. Press **Tab**.
10. Enter **20**.
11. Press **Tab**.

12. Enter **35**.
13. Press **Tab**.
14. Enter **42**.
15. Press **Tab**.
16. Continue entering **values**, using the **Tab** key to move between cells.
17. Click **Apply** when finished.

Resultset 1

recipe_ndx	Flavor	Ice	Milk	Sugar
1	Vanilla	20	35	42
2	Chocolate	30	40	50
3	Strawberry	40	50	60
4	Hokey Pokey	50	60	70

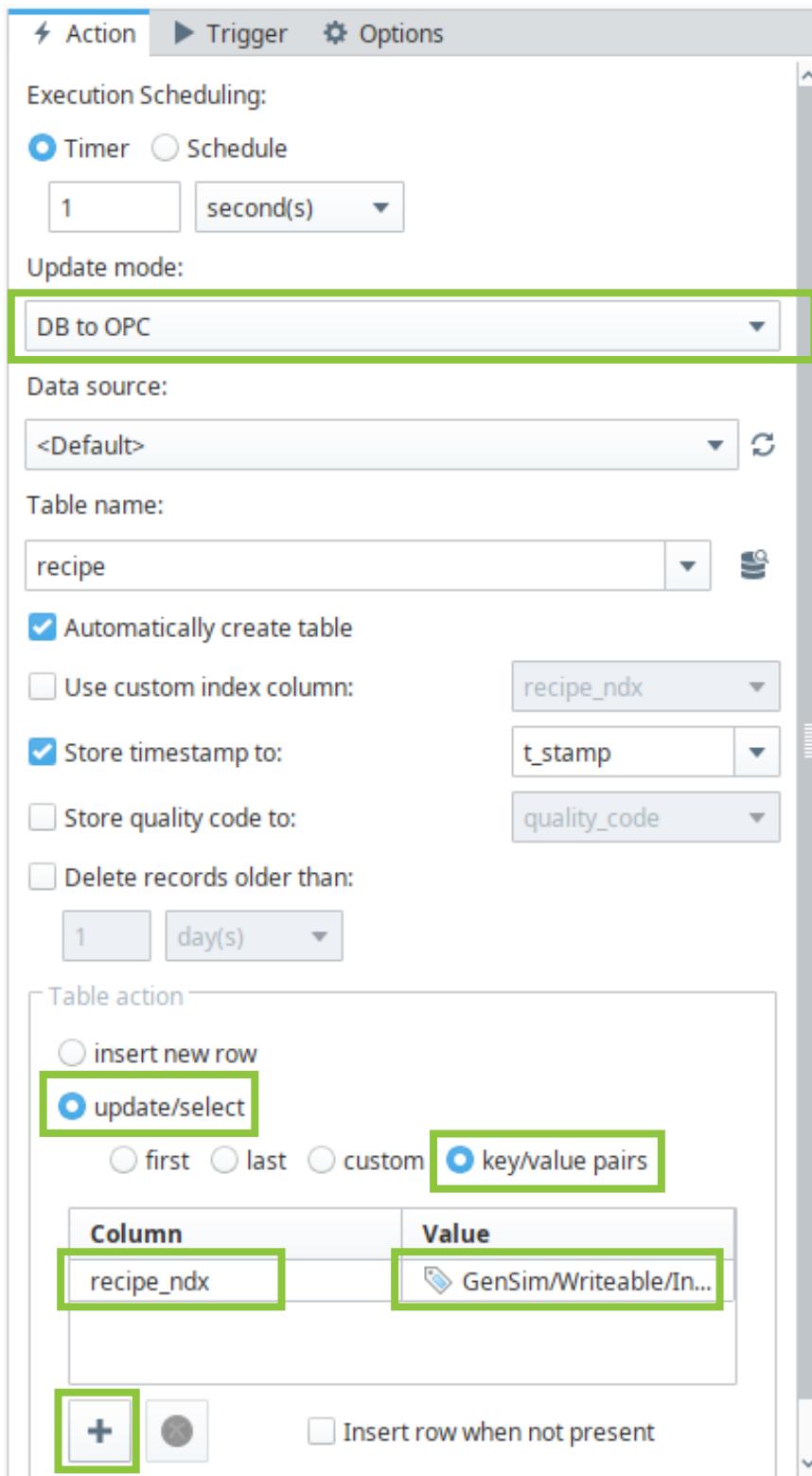
Change the Group Settings

Now that we have our database set up, we can change the Transaction Group to be bidirectional.

1. Select **DB to OPC** from the **Update mode** list.
2. Click **update/select** in the Table action.

We need to establish a relationship between the database table and the Tags. We can do this by assigning key/value pairs, which will allow us to associate the Flavor column in the database (the key) with the Flavor Tag (the value). This way, when we change the value of the Tag, the Transaction Group will be able to match that value to the associated row of the database, and assign values to the other Tags based on that row.

3. Click **key/value pairs**.
4. Click **+**.
5. Click in the **first cell** under **Column**.
6. Select **recipe_ndx** from the list.
7. Click the **first cell** under **Value**.
8. Select **GenSim/Writeable/Index** from the list.



Test the Group

Now that we have things set up, it's time to test. In order to see everything happening at once, be sure you have your **Database Query Browser** open and visible on the screen, and be sure that you can see the renamed Tags in the **Tag Browser**.

1. Click **Enabled**.
2. Click **File**.
3. Click **Save All**.

Note: When the group is running, you will see Total Executions and DB Writes incrementing, but no data will be actually written to the database because the mode is set to read from the DB instead.

4. Double-click the **value** of the **Index Tag** in the Tag Browser.
5. Enter the **value of the recipe_ndx** that matches a recipe you entered in the database.

For example, in the images above, the index for Vanilla is 1.

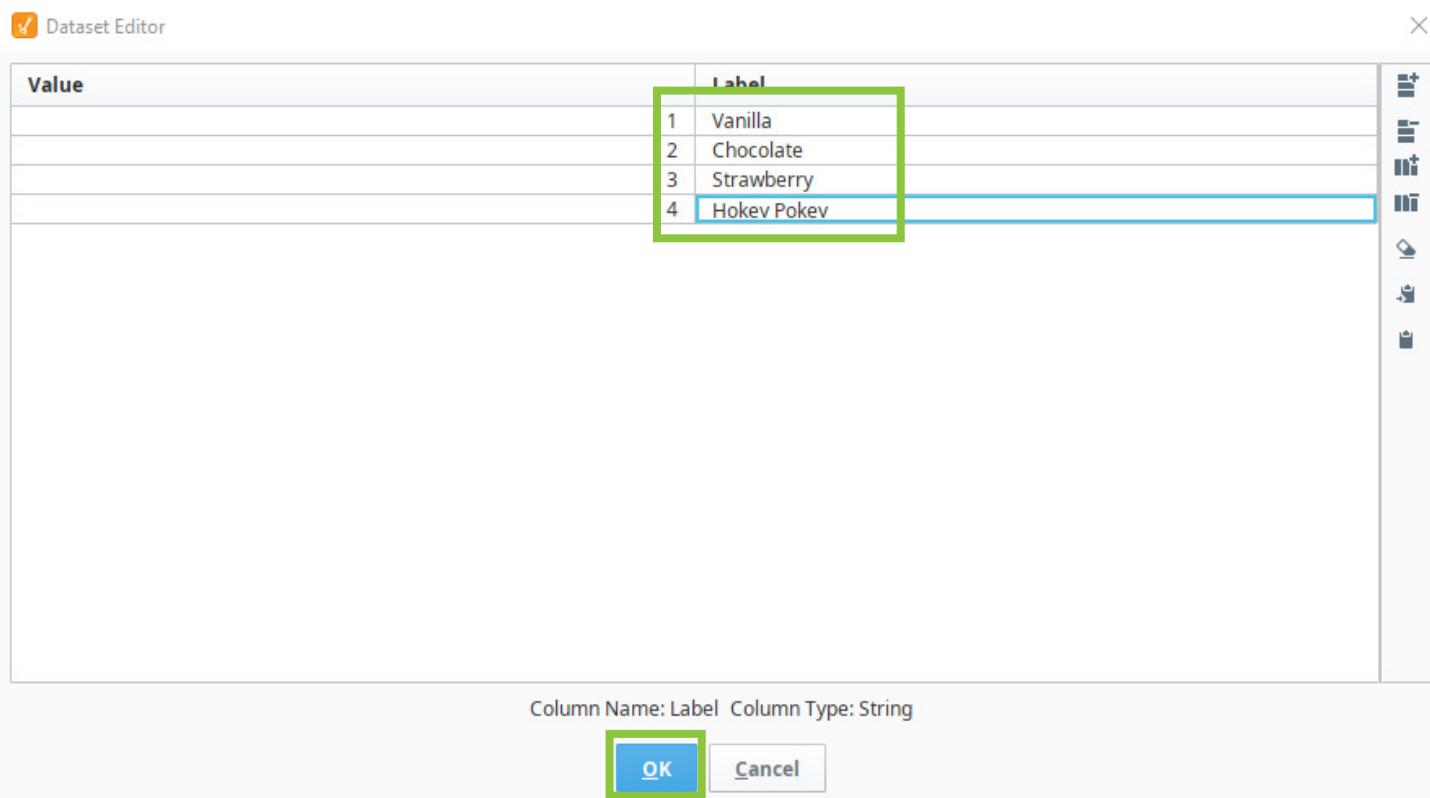
6. Press **Enter**.
7. The **Ice**, **Milk**, and **Sugar** Tag values will update to match the values in the database.

Create an Window

As a final step, let's create a Window in Vision that would allow operators to load recipes.

1. Expand **Vision** in the Project Browser.
2. Expand **Windows**.
3. Expand **Main Windows**.
4. Right-click **Empty**.
5. Click **Duplicate**.
6. Right-click **Empty(1)**.
7. Click **Rename**.
8. Enter **Recipes**.
9. Update the **header text**.
10. Expand the **Components Palette**.
11. Enter **Dropdown** in the Filter box.
12. Drag a **List** to the Window.

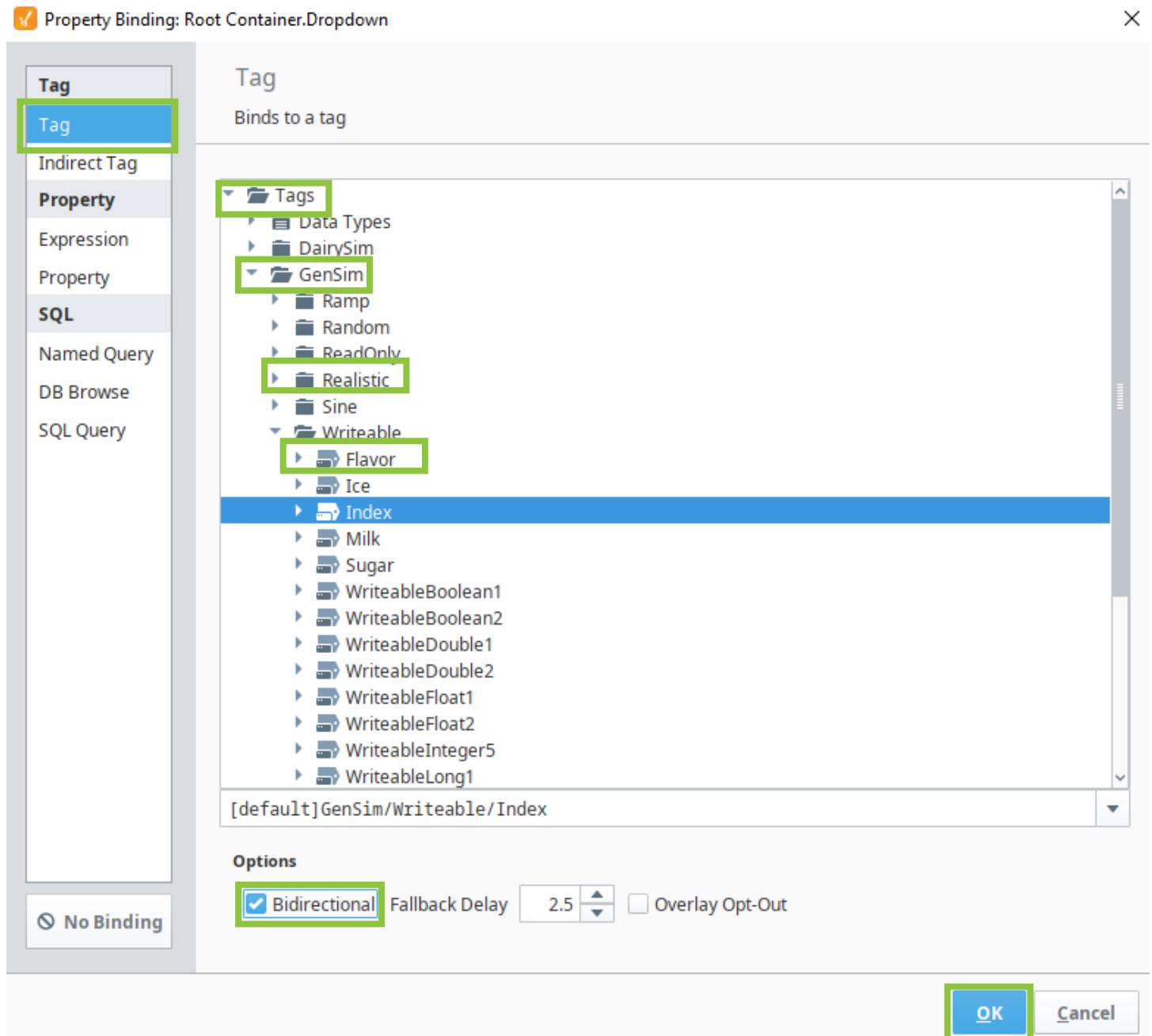
13. Click  for the **Data** property.
14. Click  four times.
15. Enter **1** in the first Value field.
16. Press **Tab**.
17. Enter **Vanilla** in the first Label field.
18. Press **Tab**.
19. Continue filling out the table with your flavors.
20. Click **OK**.



Note: Be sure to add the flavors in the exact same order that they exist in the table. Also, it would be possible to write a query and then populate the list with the data from the database, but that is beyond the scope of this course.

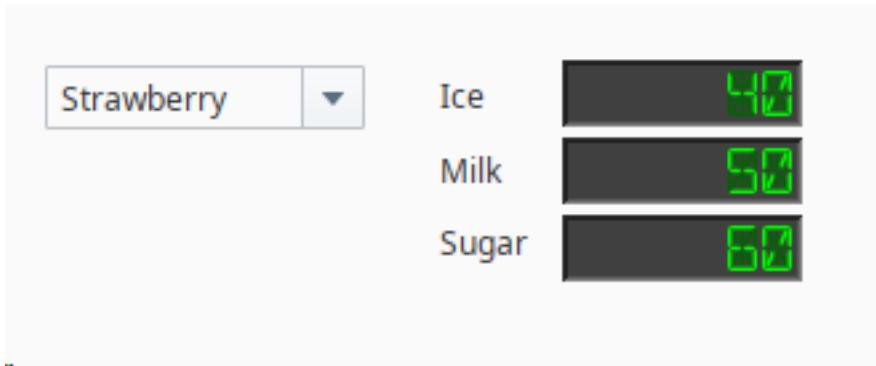
21. Click  for the **Selected Value** property.
22. Click **Tag**.
23. Expand **Tags**.

24. Expand **GenSim**.
25. Expand **Writable**.
26. Click **Index**.
27. Click **Bidirectional**.
28. Click **OK**.



29. Click and drag **Ice** from the Tag Browser to the Window.
30. Select **Display**.

31. Click **LED Display**.
32. Repeat the last three sets to add LED Display components for **Milk** and **Sugar**.
33. Add **Labels** to all four controls.
34. Click .
35. Select a **flavor** from the list.



While this is obviously a very simple example, it shows the power of Transaction Groups to both write Tag data to the database and read from the database to write data back to Tags.

Security

Ignition uses a roles-based security scheme to lock down Gateway and project resources. In effect, this means that users within Ignition are assigned roles, and their ability to access resources is dependent on the roles they are assigned. This abstraction allows users to be reassigned, removed, and added without affecting the logic of the security policy.

What We Can Secure

Security can be applied to many aspects of our Gateway and projects.

Specifically, we can use security settings to:

- Determine who can change configuration, in both the Gateway and Designer.
- Who has access to the Vision Client or Perspective Session.
- Who can read from and write to Tags.
- Who can view or interact with individual components, containers, or Windows in Vision.
- Who can view or interact with individual Views in Perspective.
- Who can execute scripts.

Gateway Communication

It is possible to require the use of Secure Sockets Layer, or SSL, to encrypt the network communication between the Gateway and Designer, between the Gateway and the Vision Client, and between the Gateway and the web browser for Perspective Sessions.

SSL cannot be used on communication between the Gateway and PLCs, but OPC-UA provides its own security protocols to encrypt this communication.

Setting up and configuring SSL and OPC-UA security is beyond the scope of this course.

Authentication Profiles

The users and their roles are defined in authentication profiles. An Ignition Gateway can have many different authentication profiles defined, each governing the security of different aspects of Gateway. For example, logging into the Gateway configuration web interface might be governed by one authentication profile, while another profile governs the security for a project.

Authentication profiles can be managed using one of six systems:

Active Directory Microsoft's industry-standard user authentication system. If you work for a large organization, there is a high probability that your company already uses AD for its IT security. If so, Ignition can simply authenticate against that, eliminating the need to manually recreate users and roles.

AD/Database Hybrid Sometimes, organizations will use AD, but the roles set up do not match the roles needed for Ignition. In that scenario, you can use AD for users, but a separate, manually-created database system to manage the roles.

AD/Internal Hybrid The same as above, but rather than storing roles in an external database, they can be stored internally within Ignition.

Database If your organization does not use AD, you can create users and roles in a set of database tables. This requires manual setup in the database. We use this setup in our advanced classes.

Fallback Cache Remote installations can connect to another Gateway for authentication, but use this as an option if network communication to the main Gateway is lost. Existing users will continue to be able to log in and work in Ignition until communication is restored.

Internal Essentially the same as the database option above, but the information is stored locally within the Gateway. This is the system used by the default authentication profile, and it is what we will be using in this class.

Add Users and Roles

As stated above, Ignition's security is all roles-based. Usernames, passwords, roles, and other data are all stored in authentication profiles.

On installation, Ignition creates an authentication profile called **default**. It contains the admin account that we have been using throughout the course. (You might remember that we created this user account as part of the installation process.) This account is in the Administrator role, which is also created on installation.

Add New Roles

We need additional users and roles so that we can explore security settings.

1. Open the **Gateway web page**.
2. Click **Config**.
3. **Log in**.
4. Click **Users, Roles** in the Security section.



5. Click **More** to the right of **default**.
6. Click **manage users**.

Name	Type	Description	
default	Internal	This user source profile was automatically created during a clean startup	More edit
opcua-module	Internal	OPC UA clients will authenticate against this profile by default.	manage users delete

[→ Create new User Source...](#)
[→ Verify a User Source...](#)

7. Click **Roles**.
8. Click **Add Role**.

The screenshot shows a light gray interface with two tabs at the top: 'Users' and 'Roles'. The 'Roles' tab is highlighted with a green box. Below the tabs, the word 'Role' is followed by a list containing 'Administrator'. At the bottom left, there is a button with a blue arrow pointing right and the text 'Add Role'.

9. Type **Operator** in the Role Name field.

10. Click **Add Role**.

The screenshot shows a 'Role Properties' dialog box. At the top, it says 'Role Properties'. Below that, there's a 'Role Name' field containing the word 'Operator', which is highlighted with a green rectangular border. To the right of the field is a blue 'Add Role' button, also highlighted with a green border. On the left side of the dialog, there's a link '[Cancel](#)'.

The new role is created, and you are returned to the previous screen.

11. Click **Add Role**.

12. Type **Manager** in the Role Name field.

13. Click **Add Role**.

We now have three roles to work with.

Add New Users

Now that we have our roles created, let's add some additional users.

1. Click **Users**.

2. Click **Add User**.

The screenshot shows an 'Add User' dialog box. At the top, there are two tabs: 'Users' (which is highlighted with a green border) and 'Roles'. Below the tabs, there is a 'Username' field containing the text 'admin'. At the bottom of the dialog is a blue 'Add User' button, which is also highlighted with a green border.

3. Type **oper** in the Username field.
4. Type **password** in the Password field.

Note: For ease of testing, we will use “password” for all users. For obvious reasons, you should make sure that all users in your system have strong passwords. Because of our weak password, you may get warnings in the browser. Simply click to ignore the warning.

5. Retype the password.
6. Click **Operator** from the list of roles.

The screenshot shows a user interface for managing user properties. At the top, there are two tabs: "Users" and "Roles". Below the tabs, the "User Properties" section is displayed. The "Username" field contains "oper" and is highlighted with a green border. The "Password" field contains "....." and is also highlighted with a green border. A second "Password" field below it also contains ".....". A note below the second password field says "Re-type password for verification." The "First Name" and "Last Name" fields are empty. In the "Roles" section, there are three checkboxes: "Administrator" (unchecked), "Manager" (unchecked), and "Operator" (checked and highlighted with a green border). The entire "User Properties" section is enclosed in a dark grey header bar.

User Properties		
Username	oper	
Password	
Password	
Re-type password for verification.		
First Name		
Last Name		
Roles	<input type="checkbox"/> Administrator <input type="checkbox"/> Manager	<input checked="" type="checkbox"/> Operator

7. Scroll down.
8. Click **Add User**.

The user is added, and you are returned to the previous screen.

9. Click **Add User**.
10. Type **manager** in the Username field.
11. Type **password** in the Password field.
12. Retype the password.
13. Click **Manager** from the list of roles.
14. Click **Save Changes**.

Our new user is added.

15. Click **Add User**.
16. Type **supervisor** in the Username field.
17. Type **password** in the Password field.
18. Retype the password.
19. Click **Administrator** and **Manager** from the list of roles.
20. Click **Add User**.

The new user is adding into both roles.

21. Click **Add User**.
22. Type **guest** in the Username field.
23. Type **password** in the Password field.
24. Retype the password.
25. Ensure that **no roles** are selected.
26. Click **Add User**.
27. Confirm that your list of users matches those shown below, and that the users are in the correct roles.

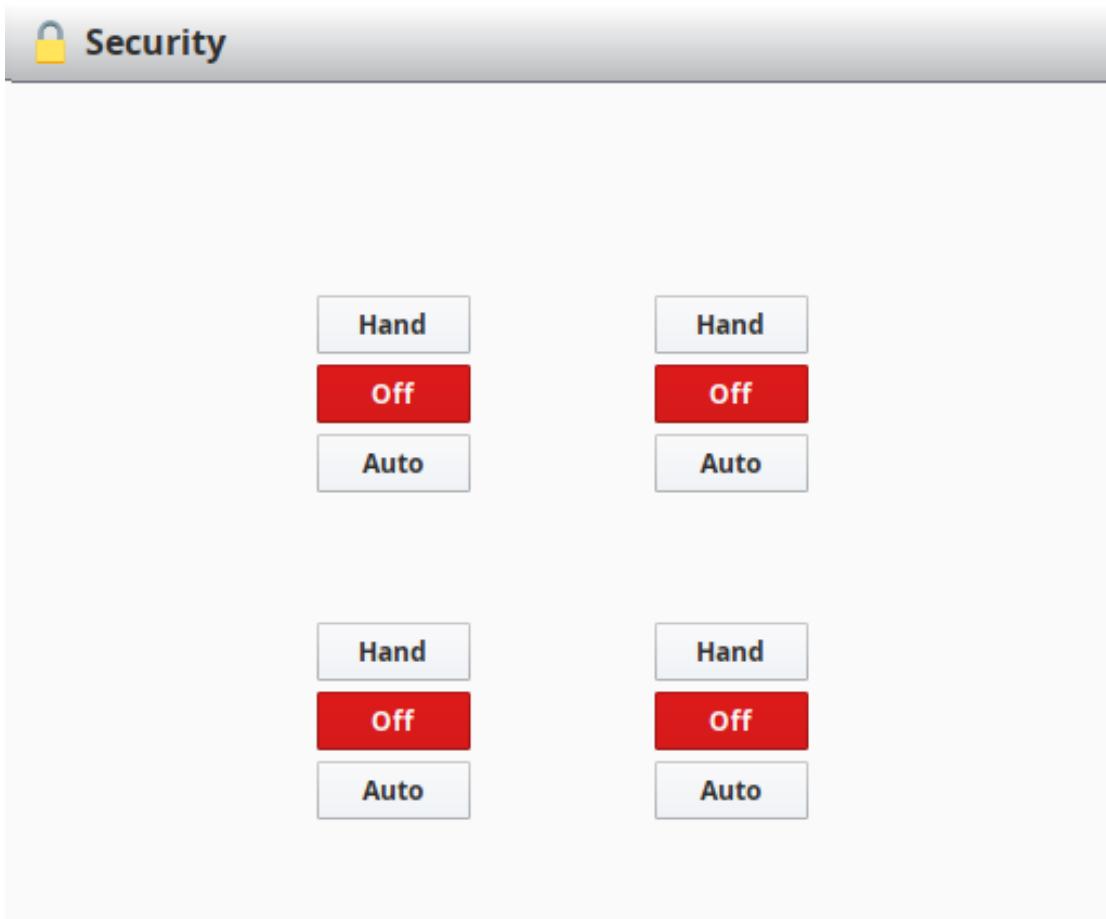
Users	Roles	
Username	Name	Roles
admin		Administrator
guest		
manager		Manager
oper		Operator
supervisor		Manager, Administrator

Component Security

Now that we have some users to work with, we want to explore options for applying security settings in our Vision project. The first set of options we will look at revolves around restricting access to individual components.

1. Return to **Designer**.
2. Expand **Vision**.

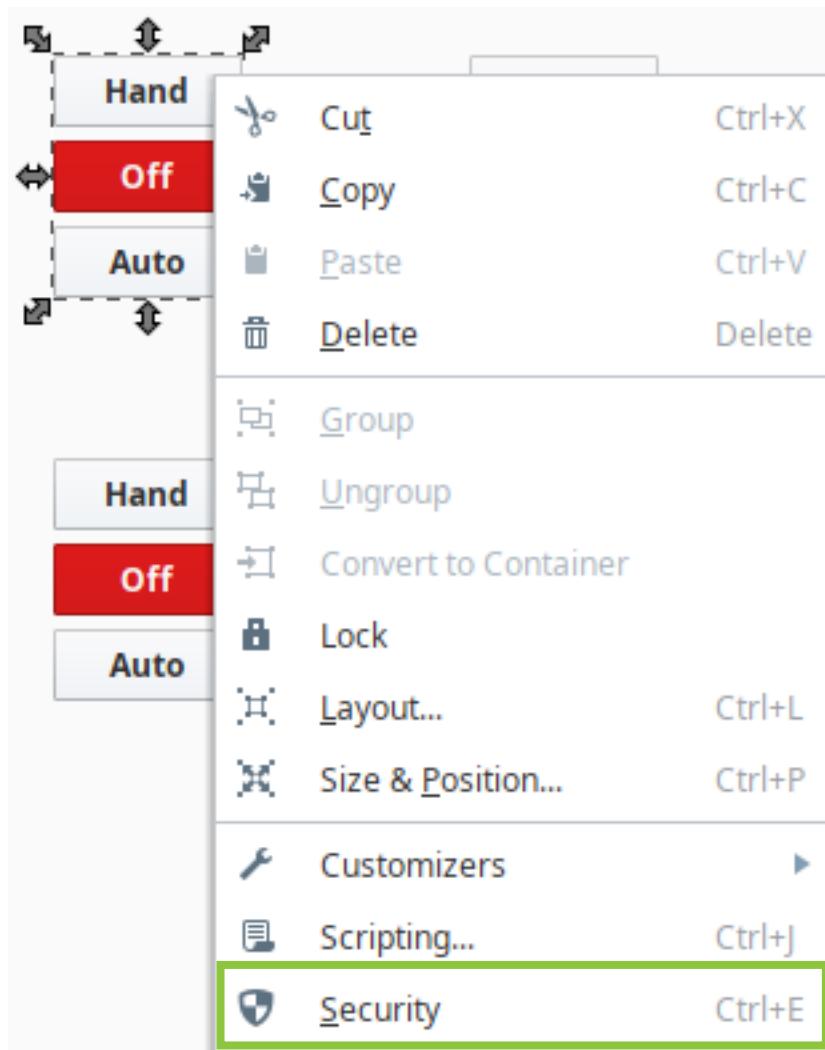
3. Expand **Windows**.
4. Expand **Main Windows**.
5. Right-click **Empty**.
6. Click **Duplicate**.
7. Right-click **Empty (1)**.
8. Click **Rename**.
9. Type **Security**.
10. Change the heading text and, if desired, the icon.
11. Expand **Components Palette**.
12. Click in the filter box.
13. Type **multi**
14. Drag a **Multi-State Button** to the Window.
15. Drag three more **Multi-State Buttons** to the Window.
16. Arrange the four buttons in the 2x2 grid.



We will use these four buttons to explore component security. We are using multi-state buttons because they offer a basic level of interactivity, however, we will not be binding them to Tags.

17. Right-click the **top left button**.

18. Click **Security**.



The Security Settings panel will open on the left side of the screen.

Component Security Options

The Security Settings panel in Designer gives us three groups of options for applying security to components.

The first is deciding whether or not this component should inherit its permissions. All components in Vision exist within a hierarchy. In this example, the buttons are directly on the window, so they are all children of the window's Root Container, and thus inherit

their permissions from that. It is possible to group components, thus creating a new parent container. This would be the preferred method for applying the same security settings to several components: rather than applying them individually, you could group them, then apply the security to the group container. Because all components are set to inherit permissions by default, applying permissions to the container would automatically apply those to all components in the group.

The second section, Restrictions, allows us to apply one or more of the following restrictions.

Access Denied Overlay Shows an overlay on top of the component when the user doesn't have security clearance.

Disable Sets the Enabled property to false on the component when the window opens up.

Disable Events Prevents event scripts from running when the user doesn't have security clearance.

Hide Sets the Visible property to false on the component when the window opens up.

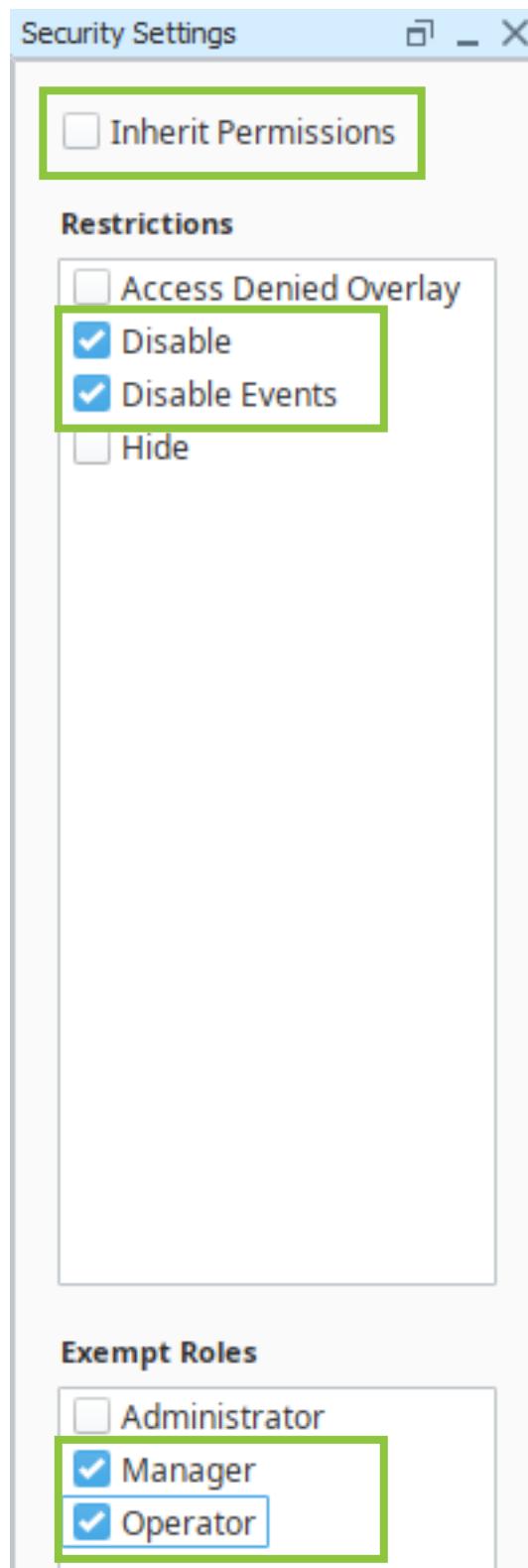
The third section of the panel, Exempt Roles, allows us to exempt roles from the restrictions applied in the second panel. You can exempt any roles from the project's user source. See "Project Security Settings" on page 250 for more details on setting the project's user source.

Note: If you do not see the roles you created in the Gateway, right-click in the Roles section of the Security panel and click Refresh. If the roles still do not appear, verify that you set up the roles in the correct user source.

Disable a Component

For our first button, we want to disable the component and any events that might be on it for our Administrator users, but still allow the use of the button for those in the Manager and Operator roles.

1. Uncheck **Inherit Permissions** in the Security Settings panel.
2. Leave **Disable** and **Disable Events** checked in the Restrictions section.
3. Check **Manager** and **Operator** in the Exempt Roles section.



A Security icon is displayed next to the component in the Project Browser, indicating that the security is set up.



Apply the Access Denied Overlay

Next, we want to apply the Access Denied Overlay to the second button for users in the Administrator and Operator roles, while allowing Manager users access to it. You should still have the Security Settings panel open from the prior steps.

1. Click the **top-right** button to select it.
2. Uncheck **Inherit Permissions** in the Security panel.
3. Check **Access Denied Overlay** in the Restrictions section.
4. Check **Administrator** and **Operator** in the Exempt Roles section.

Once again, a Security icon is displayed next to the component in the Project Browser, indicating that the security is set up.

Hide a Component

We want to hide the third button so that it does not appear at all for anyone except those in the Administrator and Manager roles.

1. Click the **bottom-right** button to select it.
2. Uncheck **Inherit Permissions** in the Security panel.
3. Check **Hide** in the Restrictions section.
4. Check **Administrator** and **Manager** in the Exempt Roles section.

Once again, a Security icon is displayed next to the component in the Project Browser, indicating that the security is set up.

We are going to leave the fourth and final button with its default security settings, which means that it will inherit permissions from its parent and in effect, have no specific security settings applied to it.

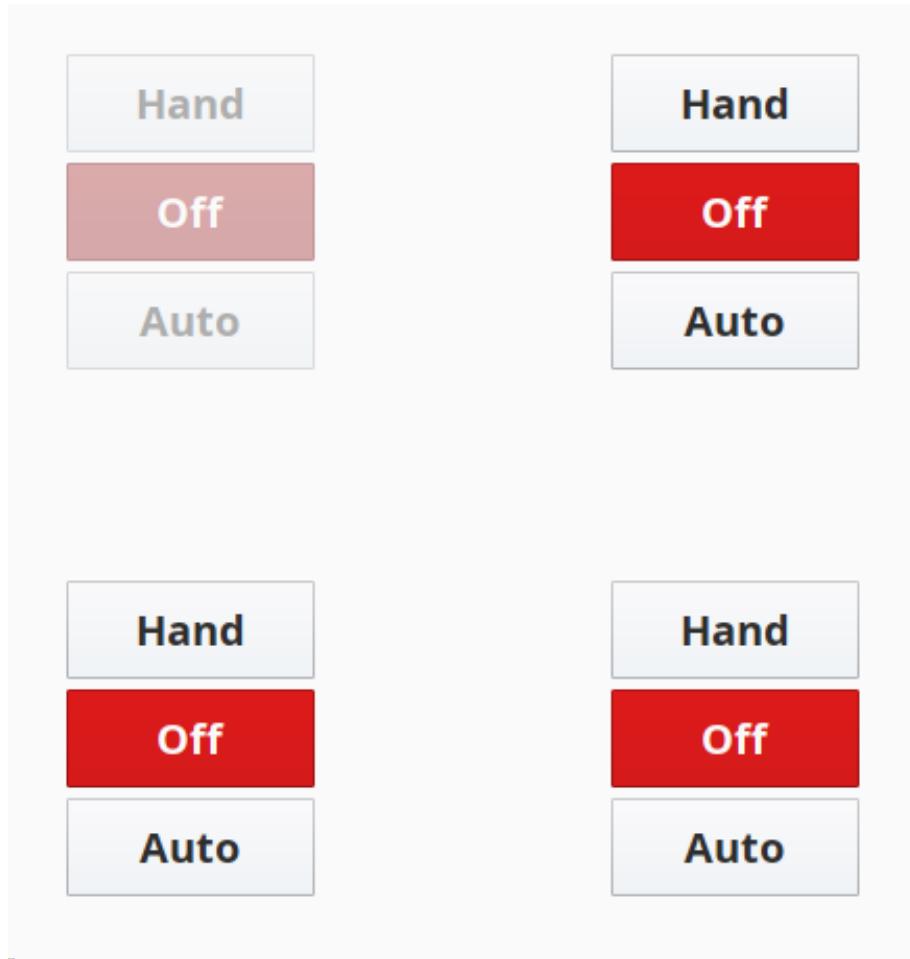
Testing Component Security

In order to test component-level security, we need to look at the project in the Vision Client. Not all of our roles have Designer access, so there is no easy way to switch between the user accounts in Designer.

Before we can do that, we need to add our new Security window to the navigation. Then, we will launch the client and test our security.

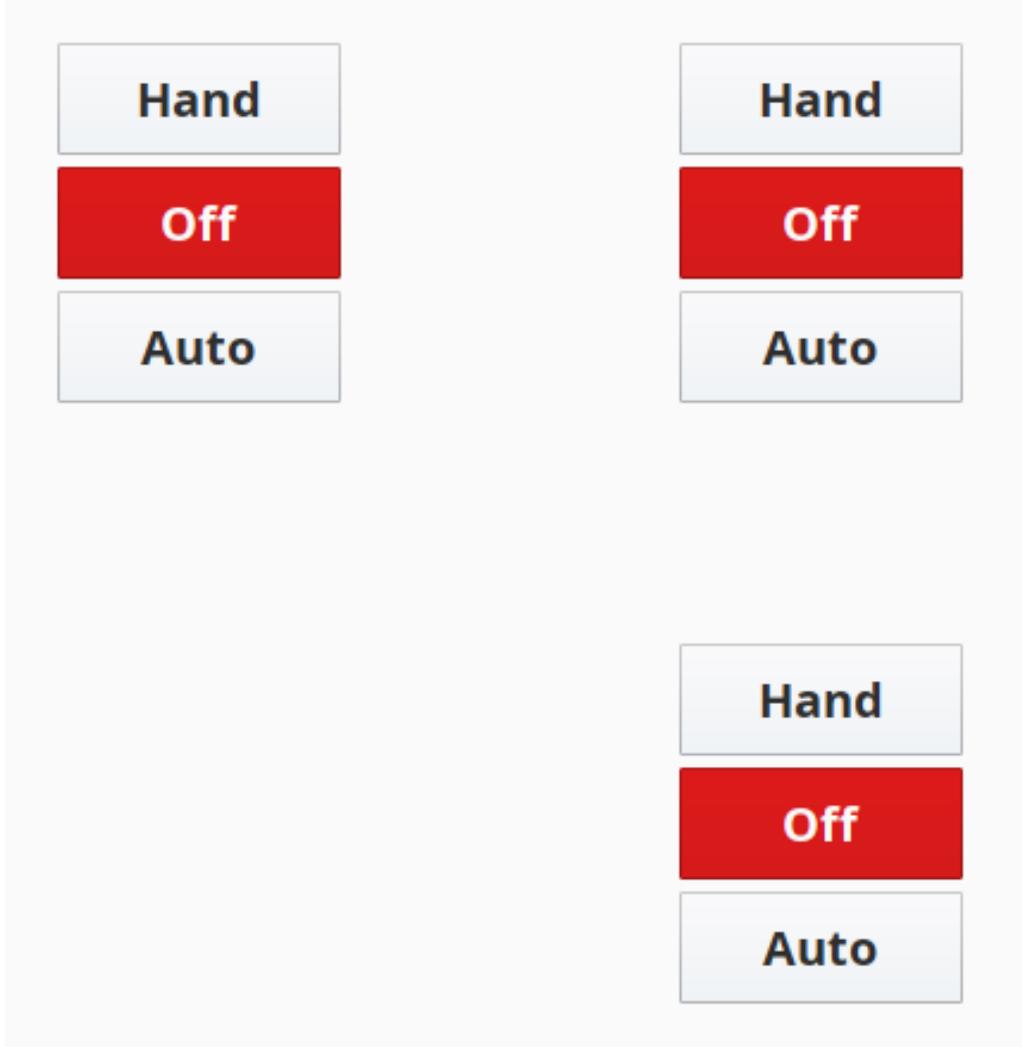
1. Double-click **Navigation** in the Project Browser.

2. Double-click the **Tab Strip** component.
3. Click the tab you want the new tab to be next to.
4. Click **Add Tab**.
5. Select **Main Windows/Security** from the Window list.
6. Type **Security** in the Name field.
7. Click **OK**.
8. Click **File**.
9. Click **Save All**.
10. Click **Tools**.
11. Select **Launch Project**.
12. Click **Launch (windowed)**.
13. Log into the client with the **admin** account.
14. Click **Security**.



Because we are logged in as admin, which is in the Administrator group, the top-left button will be disabled, and the top-right button will show the Access Denied Overlay, and the two bottom buttons will be fully enabled.

15. Click **Switch User**.
16. Type **oper** in the Username field.
17. Type **password** in the Password field.
18. Click **OK**.



Our oper user is in the Operator role. Therefore, the top-right button will show the Access Denied Overlay, and the bottom-left button will be hidden.

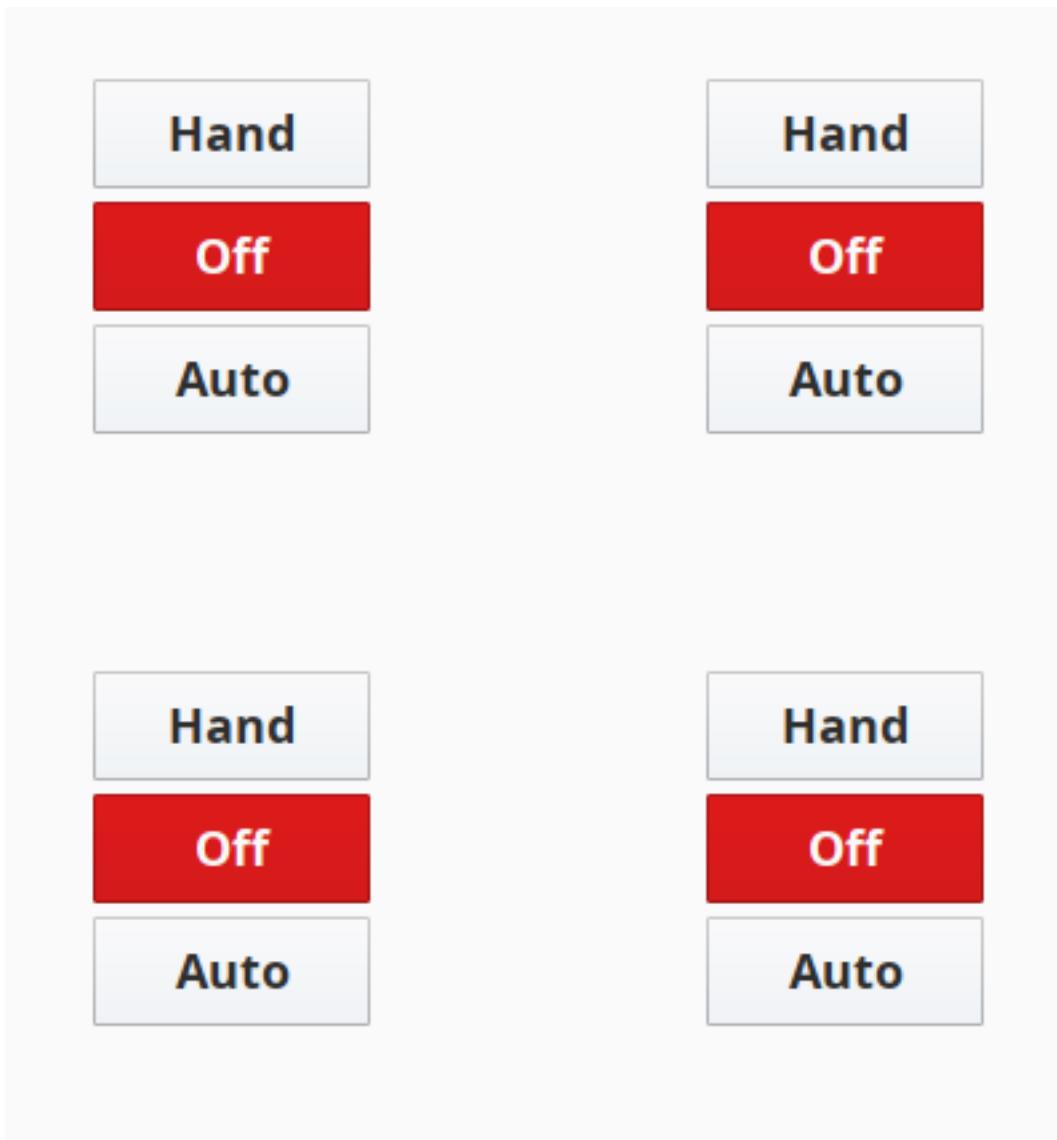
Note: There is no easy way to customize the appearance of the Access Denied Overlay. It can only be done by using the Ignition SDK and writing Java code.

Users in Multiple Roles

Our supervisor user is in both the Administrator and the Manager role, while our manager user is both a Manager and an Operator.

Our buttons have conflicting permissions for these role combinations. For example, the top-left button is disabled for the Administrator role, but enabled for the Manager role. We want to explore what happens when users exist in more than one role.

1. Click **Switch User**.
2. Type **supervisor** in the Username field.
3. Type **password** in the Password field.
4. Click **OK**.

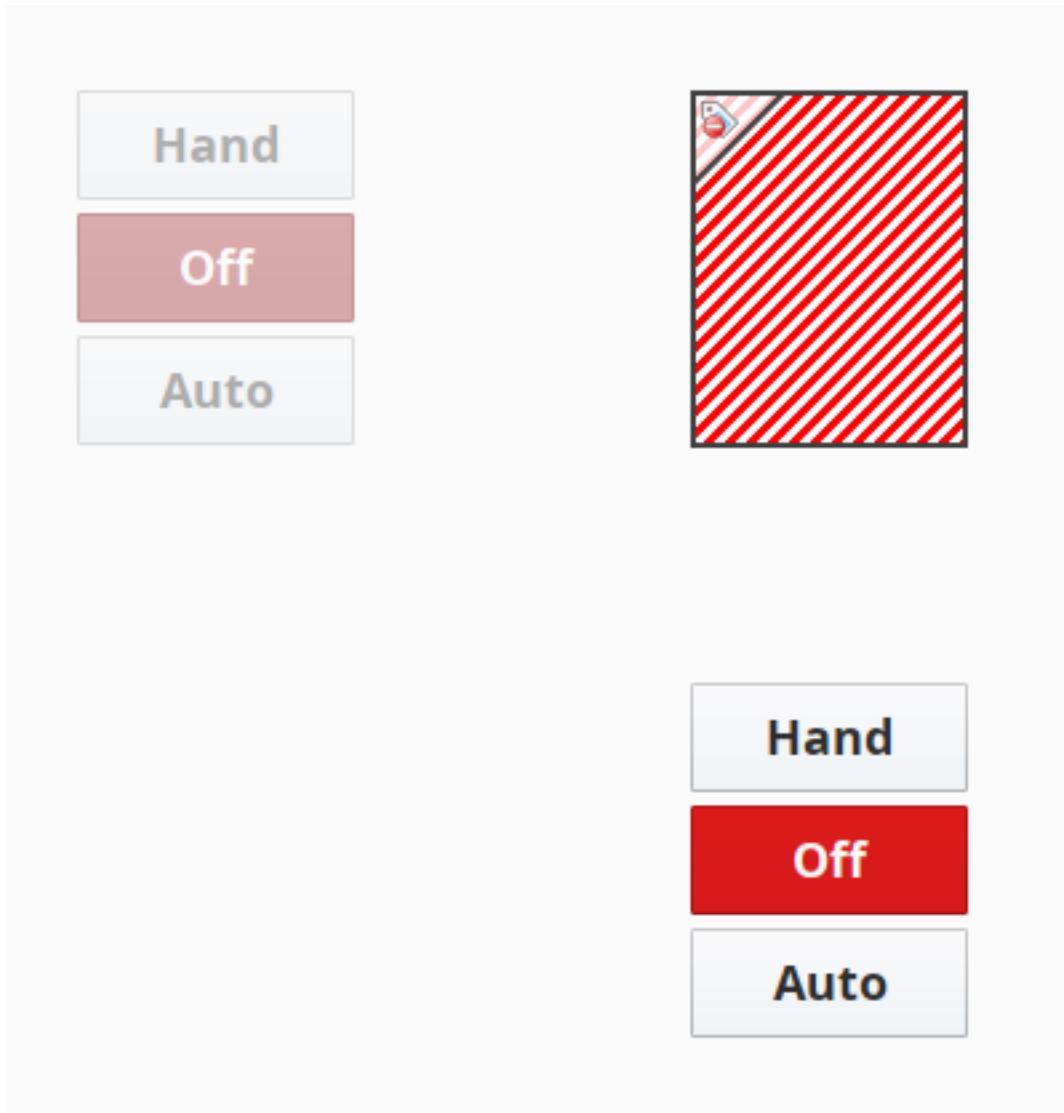


You will see here the result: users in multiple roles are provided with the least restrictive combination of the security settings for their roles. As an Administrator, the supervisor role should have the top-left button disabled, but as a Manager is should be enabled, and you'll see that the button is enabled. The same is true for the top-right button: it should show the Access Denied Overlay for Administrators but not Managers, so again, our user in both roles has access to it because one of their roles does.

Users in No Roles

For our final example, we want to take a look at our guest user, who is not in any roles.

1. Click **Switch User**.
2. Type **guest** in the Username field.
3. Type **password** in the Password field.
4. Click **OK**.



While users in multiple roles are granted the least restrictive combination of the security settings, users in no role at all will get the most restrictive settings. Any security settings that apply restrictions will automatically apply to a user in no roles.

Adding Security to a Window

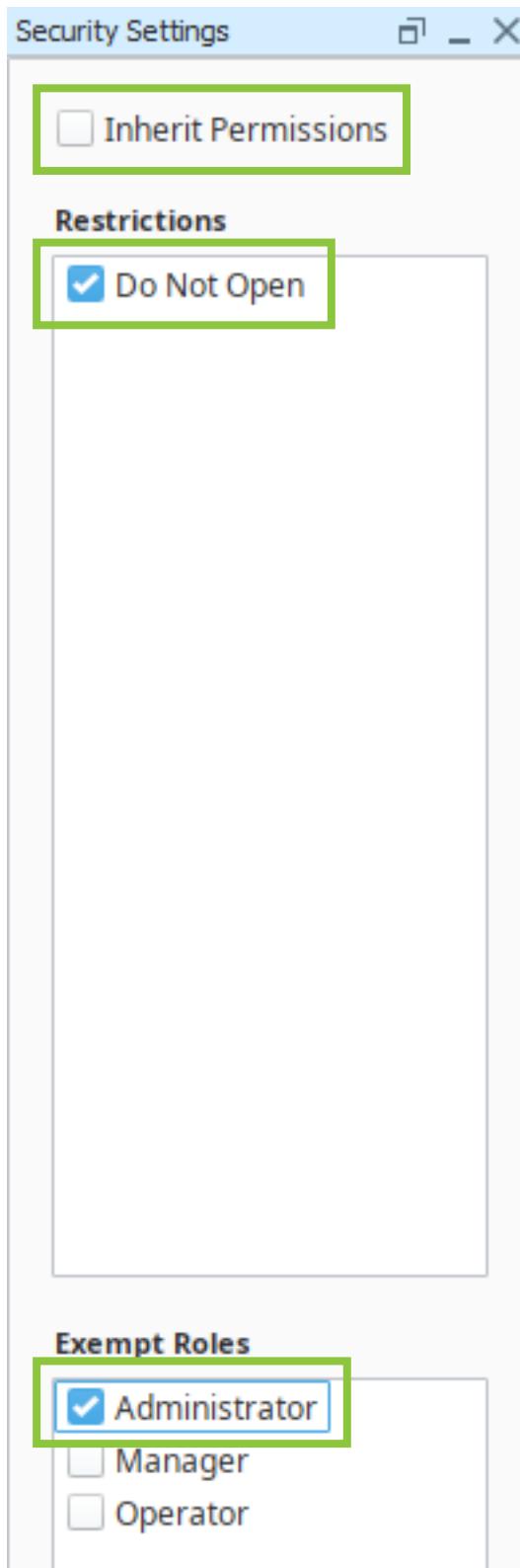
You can also apply security settings on Windows. Let's add security to our History window.

1. Double-click **Tag History** in the Project Browser.

Note: You cannot configure security properties on a Window unless it is open in Designer.

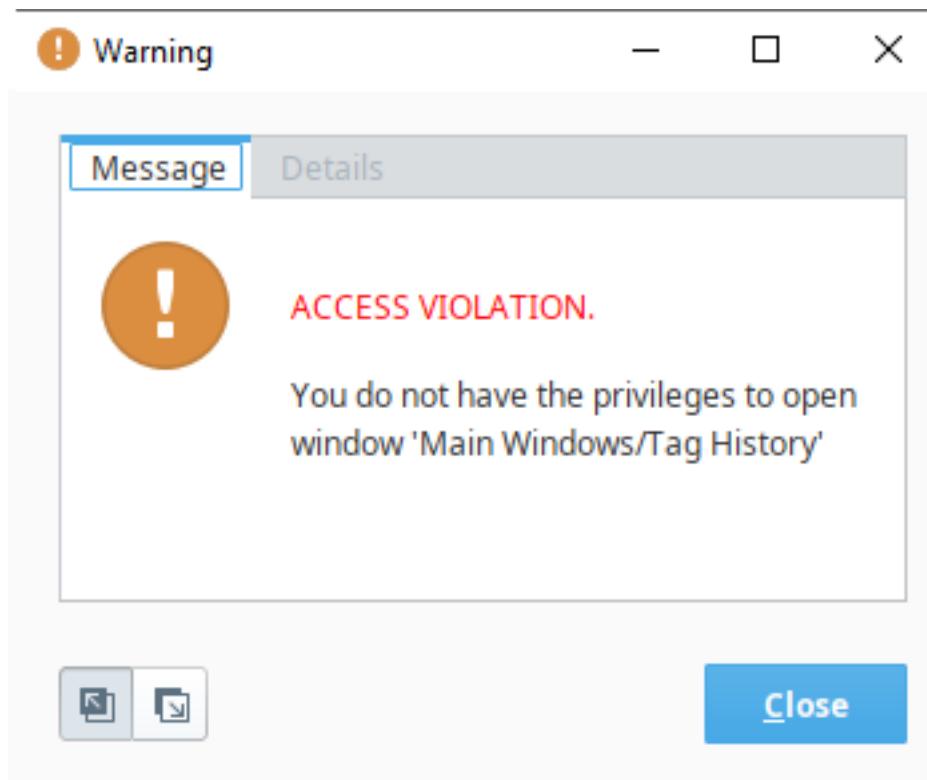
The Security panel should still be open from the prior exercise. You will see that it is laid out in the same way, with the option to inherit permissions, the restrictions, and the exempt roles. However, you will see that there is only one restriction: **Do Not Open**.

2. Uncheck **Inherit Permissions**.
3. Check **Administrator** in the Exempt Roles section.



4. Click **File**.
5. Click **Save All**.
6. Return to the **Client**.

7. Click **Tag History** on the navigation tab strip.



We are still logged in as our guest user, so we see an error when we try to access the Window. Only users in the Administrator role can view it.

1. Click **Close**.
1. Click **Switch User**.
2. Type **supervisor** in the Username field.
3. Type **password** in the Password field.
4. Click **OK**.
5. Click **Security** on the navigation tab strip.

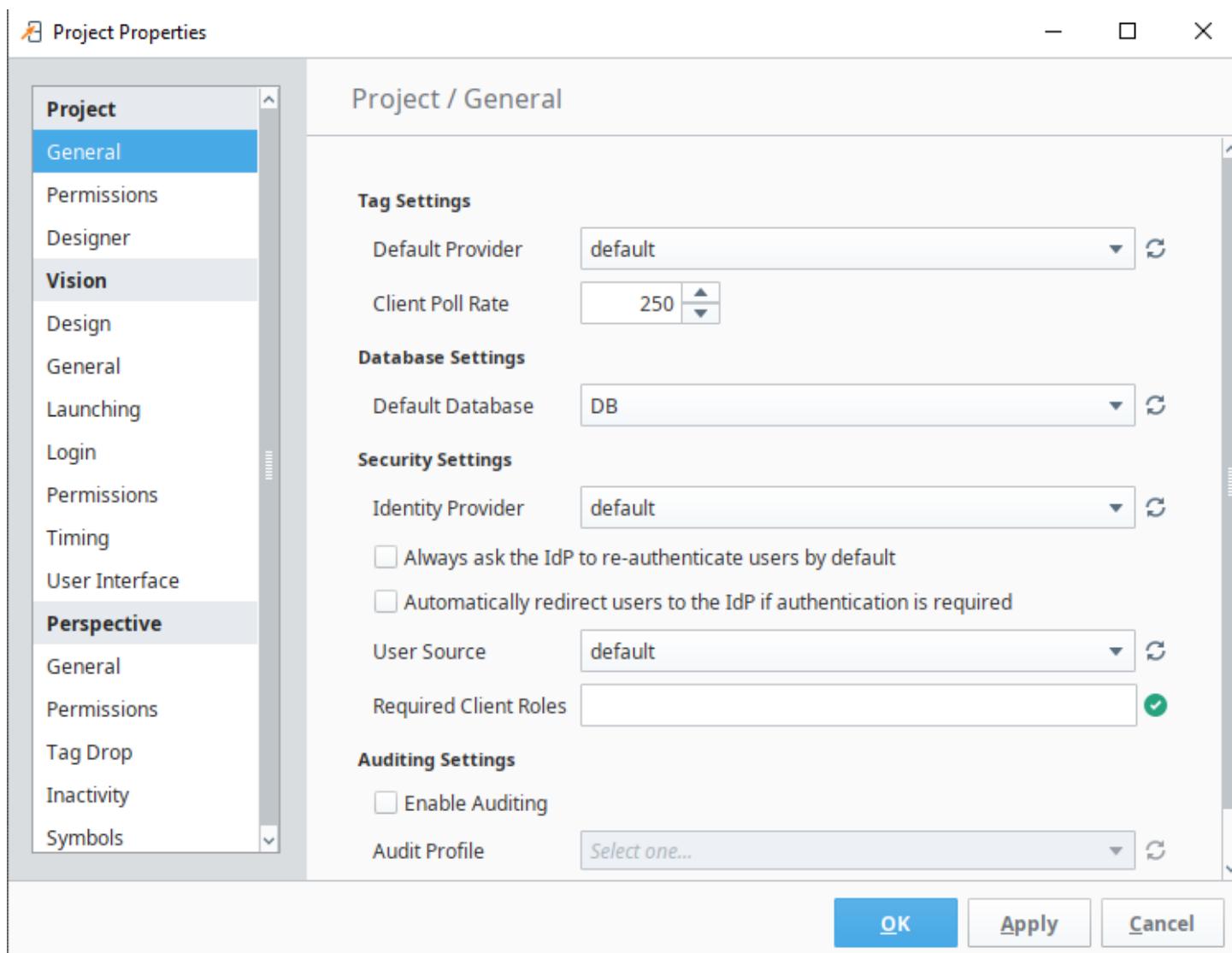
Our supervisor user is in the Administrator role, and so they do have access to the window.

Project Security Settings

In addition to component and window-based security, we can apply security settings to the project.

1. Return to **Designer**.

2. Click **Project**.
3. Click **Project Properties**.



In the Project Properties dialog box, you will see three categories for Permissions, but several other categories have security-related settings as well.

Project General

The General category of the Project has a Security section. There, you can set the Identity Provider (see "Identity Providers" on page 254 for more details), as well as the project's User Source. You can also set required roles to access the project; if roles are listed here, users not in one of those roles will be unable to log into the Client.

Project Permissions

The Permissions category in the Project section, allows you to set roles for the Designer. You can restrict who can view projects in the Designer, who can save changes, who can delete resources, and who are access project resources.

Vision Login

In the Login category of the Vision section does not directly impact security settings, but does provide options to customize the Vision Client log in screen.

Vision Permissions

The list of features in the Permissions category in the Vision section allows you to restrict access to specific Ignition subsystems. By default, all are disabled. Enabling any of these allows all users to access elements that use them, but you can also provide a comma-separated list of roles to still provide limited access.

The features are:

Alarm Management Allows the Client to cancel, shelve, and acknowledge alarms. Applies to both local and remote Alarms.

Datasource Management Allows the Client to modify Gateway datasource connections.

Device Management Allows the Client to modify device connections.

DNP3 Management Allows the Client to freeze DNP3 operations via scripting.

Legacy Database Access Allows Clients to run queries directly against the database. This doesn't effect named queries.

OPC Server Management Allows the Client to modify OPC server connections.

SFC Management Allows the Client to start or stop Sequential Function Charts.

Tag Editing Allows the Client to add, edit, or delete Tags through scripting.

Tag History Allows the Client to query/modify Tag History.

Translation Management Allows the Client to modify translations in the localization system.

User Management Allows the Client to modify schedules, holidays, and users through scripting or components.

Perspective Permissions

We will discuss Perspective permissions in detail in "Perspective Security" on page 440.

Gateway Security

The final place security is used is on the Ignition Gateway itself. While we are not going to change any of these settings, they are worth exploring.

These settings are all located in under the Security heading of the Gateway's Config section. We are not going to discuss all of the sections, as several of them are beyond the scope of this class.

General

The General category provides a series of high-level security options.

System Identity Provider This setting controls access to the Gateway's web pages and the Designer, if the Designer's authentication strategy is set to Identity Provider. See "Identity Providers" on page 254 and "Project General" on page 251.

Designer Authentication Strategy This sets how users can authenticate in Designer. Classic displays the login screen we have become familiar with, while Identity Provider uses that method. See "Identity Providers" on page 254.

System User Source This sets the default user source for Designer when using the Classic strategy.

Designer Role(s) You can determine what roles have access to Designer here. The default setting, Administrator, means that we could be using our supervisor user to log into Designer, as well as the admin user we have been.

Create Project Role(s) You can provide a list of roles that are allowed to create new projects in Designer. While it would not make sense to have a role here that can create projects but not be included in the list above to log into Designer, the opposite might be true: you might have roles that you want to give Designer access to, but only allow them to work in existing projects.

Gateway Config Permissions Throughout class, we have been using our admin user to log into the Config section of the Gateway. This is possible because the Administrator role is provided here. We could choose to allow other roles to access Config as well. Note that this uses the Security Levels setup, rather than just roles. See "Security Levels" on page 255 for more details.

Status Page Permissions The same as above, but for the Status section of the Gateway.

Home Page Permissions We have seen that the default Gateway Home Page merely provides a series of links to download and access resources. As there is nothing proprietary on the Home Page, it is set by default to allow access to anyone. However, it is possible to change the default Home Page, and if you do, it might make sense to limit access to it. Like the two settings above, this uses Security Levels.

User Inactivity Timeout You will have noticed that anytime you are away from the Gateway web page for some time, you need to log back in. This setting controls how long, in minutes, you can not interact with the Gateway before it times out and forces a log in. Setting this to a value of 0 (or any negative number) will mean that the Gateway never times out.

Allow User Admin Vision includes a User Management Component to allow for adding, removing, and editing users and roles in the Client. Regardless of any other settings, that component will be disabled unless this setting is enabled.

Allow Designer SSO If you want to use your organization's single-sign-on authentication, you need to enable this.

Gateway Audit Profile You can provide an audit profile to log Gateway events. Note that audit profiles are beyond the scope of this class.

Identity Providers

Even if you have never heard the term "identity provider," odds are very good you have encountered them before. Many web sites today allow users to either create a new account for the site or use their existing account from a service such as Google, Apple, or Facebook. All three of those companies, and many others like them, act as an Identity Provider (IdP). In essence, an identity provider is some kind of server that provides user authentication. Many very large organizations, like the three listed above, have hundreds of millions, or in some cases billions, of users, and allow other sites to simply use those user accounts.

While being an identity provider is not the primary business for Facebook or Google, there are also companies whose whole business is in providing user authentication. It's possible that your organization has you log into the network via a company such as Duo or Okta. These are just two examples of companies whose business model is built on being an identity provider.

Setting up a third-party identify provider is beyond the scope of this course, and mostly involves signing up with one of the companies that provides that as a service and then filling in the appropriate information in the Identity Provider section of the Gateway.

Ignition has also a built-in identity provider in the Gateway. This allows you to set the Designer or the Client to use an identity provider authentication strategy. This will still authenticate against one of the user sources in the Gateway, but will force users to log in through a browser interface instead of the native login screen we have been using in class so far.

Security Levels

With Security Levels, you define a hierarchy for access inside a Perspective Session or Vision Client. This authorization system provides a way for you to map roles from an Identity Provider (IdP) to Ignition roles. Any IdP can be used to provide roles, and security levels are independent of the type of IdP being used. Any role from the IdP is automatically granted to the user as a role, but only roles in your Security Levels are available to the security screens in the Designer. You can also use the User Grants option to grant additional access for each user.

Security levels are particularly useful in Perspective, so we will return to this topic and cover it in detail in "Perspective Security" on page 440.

Security Zones

A security zone is a group of gateways, computers, or IP addresses that share unified security settings. Though Ignition relies heavily on a role-based security model, it is possible to configure some high-level security restrictions based on security zone, thereby controlling access based on where a given resource request is coming from, rather than what user is requesting it.

As with Security Levels, we will examine Security Zone in more detail in "Perspective Security" on page 440.

Alarming

Alarming is a core feature of the Ignition platform. Alarms are conditions that are evaluated with respect to a specific numeric datapoint. Most commonly, alarms are configured on a Tag or a Transaction Group item.

Alarm Basics

Any number of alarms can be attached to a datapoint. When the alarm condition becomes true, the alarm is considered to be “active”. For example, on an analog Tag you might add a high-level alarm that goes active when the Tag’s value is over 50.0. You could add another alarm on the same Tag for a high-level when the value is over 75.0. On a discrete Tag, you might configure an alarm to be active when the value is non-zero. Each alarm has its own name, priority, and many other settings.

When the alarm condition becomes false after it has been true, the alarm is said to have “cleared”. Each alarm has a numeric deadband and time delay deadbands that can affect when the alarm goes active and clear, this is to prevent frequent oscillations of an alarm between active and clear when the value is hovering near the setpoint. At some point, an operator can acknowledge an alarm, confirming that they have seen the event.

The history of alarms is stored in any configured alarm journals. These will journal all state changes on each alarm, as well as acknowledgments, to an external SQL database of your choosing.

Alarm Properties

Every alarm you configure has a set of properties that can be set.

Alarm Name

The name of the alarm will be used to reference it throughout the rest the project.

Enabled

This boolean determines whether or not the alarm will be evaluated. A disabled alarm’s condition is not evaluated and thus does not generate any alarm events. Generally, this would be set dynamically based on some other criteria.

Priority

Can affect how it appears in an alarm status table, or can affect how it is escalated through a pipeline. There are five priorities: Diagnostic, Low, Medium, High, and Critical. Each priority also has associated integer value, counting from 0 (Diagnostic) up to 4 (Critical), that can be used to reference the priority in scripts and expressions.

Timestamp Source

Chooses where the timestamp for the alarm event should come from: the system time or when the event was generated, or the timestamp of the value that tripped the event.

Label

This string value is used to display alarm information to the operators.

Display Path

This string value is used to display the alarm to the operators. If this value is blank, the operator will see the path to the Tag instead.

Ack Mode

Dictates how acknowledgment works for the alarm.

Unused Acknowledgment is not used for this Tag. Any alarm that is generated will automatically be marked as acknowledged.

Auto The alarm is acknowledged automatically when the alarm becomes cleared.

Manual The alarm is never set to acknowledged by the system; it is up to operators to manually acknowledge alarms.

Notes

A place for any free-form documentation about the alarm that can be displayed to operators.

Ack Notes Required

If this setting is true, the operators cannot acknowledge this alarm without entering some notes.

Shelving Allowed

If this setting is true, the shelving feature is available for this alarm.

Mode

Sets the condition that the alarm is evaluating. The following are the different mode options:

Equal Active when the Tag's value equals the alarm's setpoint.

Not Equal Active when the Tag's value is 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 is 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, an event is generated for each value change outside the setpoints.

Out of range 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 Generates an alarm event 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 Active 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 alarm when the bit is low.

On Condition Use this free-form mode 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 / High Setpoint

The setpoint properties are used for many alarm modes to specify the range for which the alarm becomes active.

Inclusive / Low Inclusive / High Inclusive

The inclusive properties correspond to a setpoint. If true, the range will be active if the value is exactly equal to the setpoint, not only above or below it.

Deadband Mode

The following are the two deadband modes:

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.

Deadband

The deadband value is 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 is 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, with a deadband of 2. The alarm goes active if the value is between 50 and 70, but will only clear if the value falls below 48 or rises above 72.

Active Delay

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".

Clear Delay

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".

Ack Pipeline

The name of an alarm notification pipeline to put this alarm into when the alarm becomes acknowledged.

Active Pipeline

The name of an alarm notification pipeline to put this alarm into when it becomes active. This is the primary source for sending out alarm messages. Many alarms may share a single pipeline.

Clear Pipeline

The name of an alarm notification pipeline to put this alarm into when it becomes clear.

Custom Subject

A string that is 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 is used instead.

Custom Message

A string that is 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 is used instead.

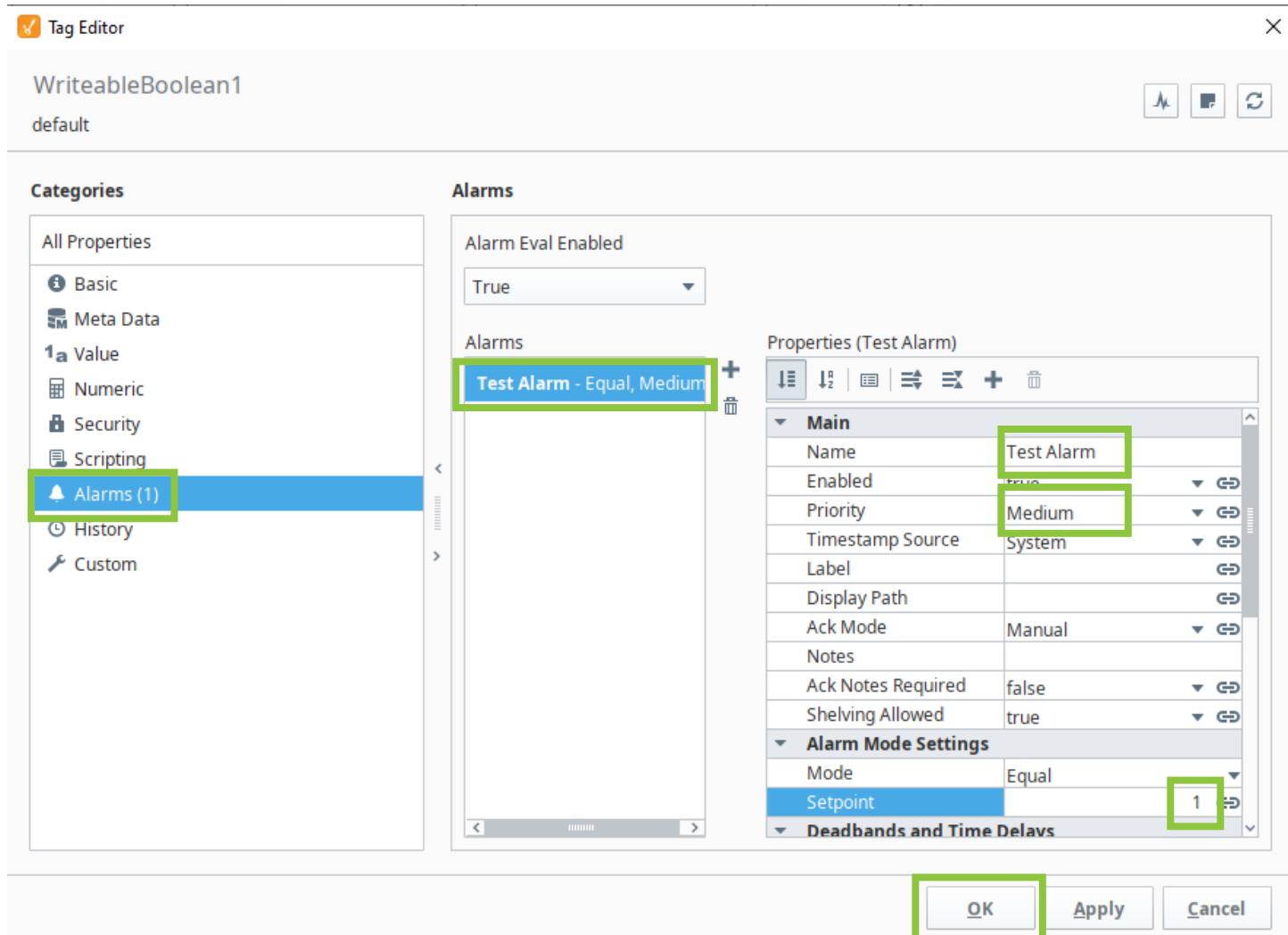
Configure an Alarm

As a first example, let's set an alarm that will be active if the value of the WriteableBoolean1 Tag is true, and inactive when it's false. Configuring an alarm on this Tag will allow us to easily control the alarm's status.

1. Expand **GenSim** In the Tag Browser.
2. Expand **Writable**.
3. Double-click **WriteableBoolean1**.
4. Click **Alarms**.
5. Double-click **New Alarm**.
6. Enter **Test Alarm** in the Name field.
7. Select **Medium** from the Priority list.
8. Double-click the **Setpoint** value.
9. Enter **1**.

10. Press **Enter**.

11. Click **OK**.



Now that the Tag has an alarm set on it, you will see a bell icon next to the Tag in the Tag Browser.

Viewing Alarms

There are several different components in Vision to display alarms.

Because we started our project using the Vision Tab Nav project template, we already have a window with one of those components, the Alarm Status Table, set up on it.

1. Expand **Vision** in the Project Browser.
2. Expand **Windows**.
3. Expand **Main Windows**.

4. Double-click **Alarms**.

The Alarm Status Table on this window should currently be empty.

Note: If you selected the option to Enable Quick Start when you first installed Ignition, you may already have alarms showing in the table. You can just ignore those.

Enabling an Alarm

In order to populate the table, let's enable the alarm we set up in the prior steps.

1. Click the checkbox next to **WriteableBoolean1**.

By toggling the Tag's value to true, you will activate the alarm. You should see it appear in the table.

2. Click the checkbox next to **WriteableBoolean1** again.

Alarms					
	Active Time	Display Path	Current State	Priority	Event Id
<input type="checkbox"/>	8/9/24, 8:39 PM	GenSim/Writeable/WriteableBoolean1/Test Alarm	Cleared, Unacknow...	Medium	02423b7a-8b04...
<input checked="" type="checkbox"/>					
<input checked="" type="checkbox"/>					

Acknowledge **Shelve**

|

Toggling the Tag to false will cause the alarm to become inactive. You will still see it on the table, but now it will indicate that it is inactive.

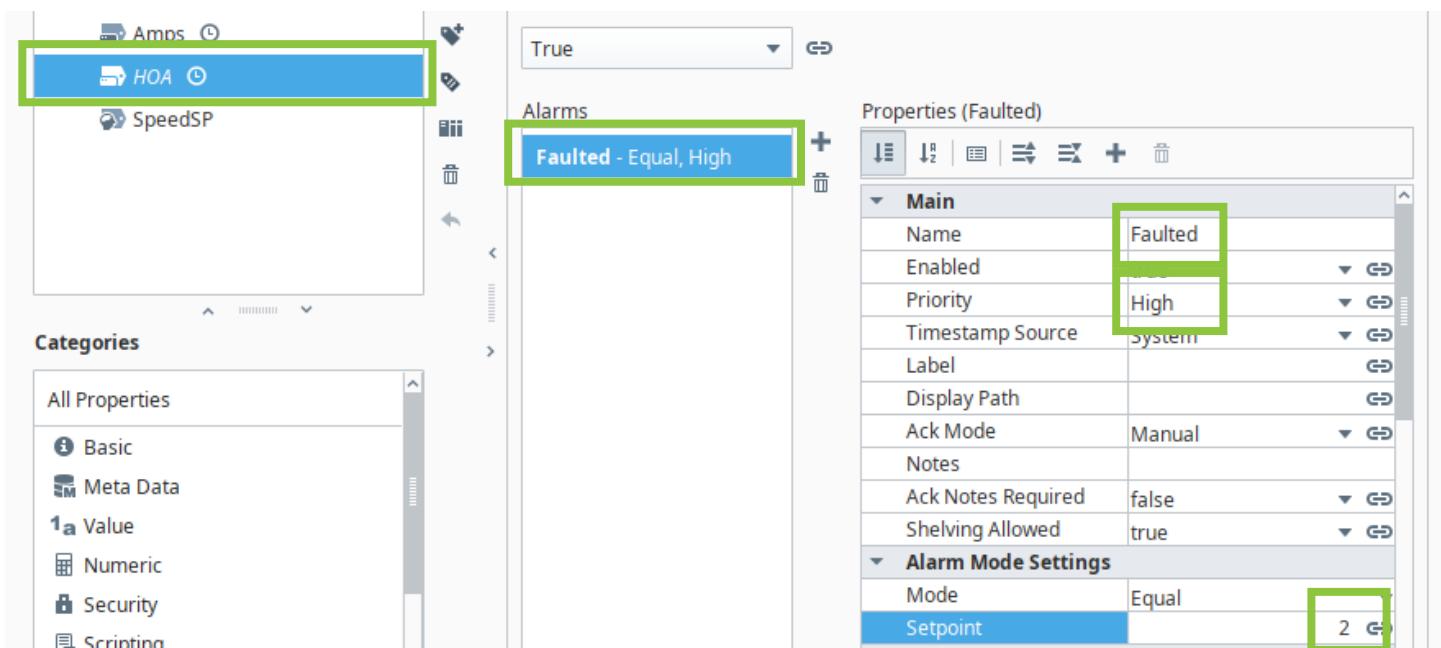
Add UDT Alarms

In order to really explore the features of the Alarm Status Table, it will be helpful to have more alarms. We also want to look at alarms that function more like real alarms, which are constantly becoming active and inactive as the Tag's value changes. We can do this by adding alarms to the UDT, which will add them to all UDT instances.

1. Click **UDT Definitions**.
2. Double-click **MotorUDT**.
3. Click **HOA**.
4. Click **Alarms**.
5. Double-click **New Alarm**.
6. Enter **Faulted** in the Name field.
7. Select **High** from the Priority list.
8. Enter **2** in the Setpoint field.

Note: Remember that a value of 2 on the HOA Tag is associated with the “Hand” stated on the linked Multi-State Buttons.

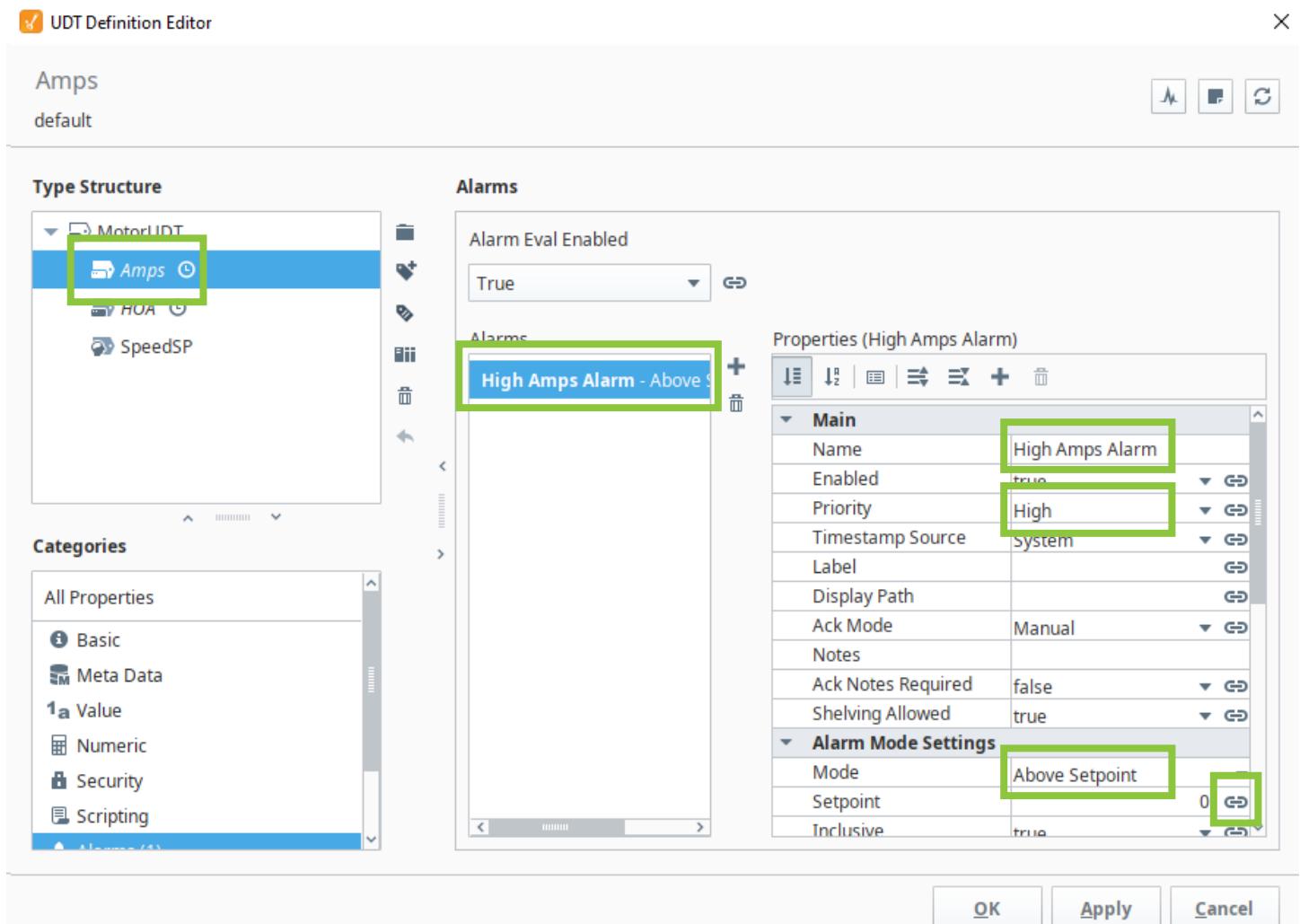
9. Leave the dialog box open for the next set of steps.



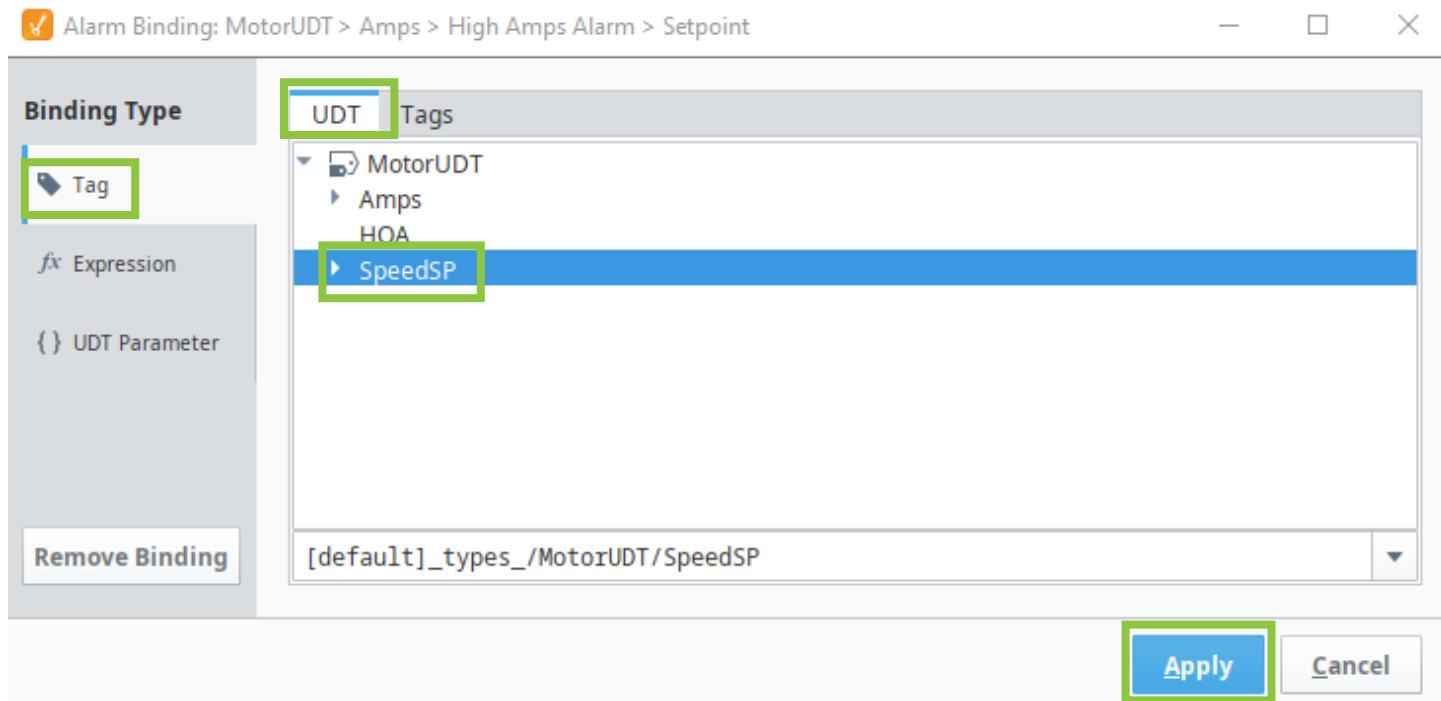
Dynamic Setpoints

We also want to set an alarm on the Amps Tag, but in this case, we want the setpoint to be dynamic. Rather than the hard-coded values we've been using, we want to have the alarm be triggered by the value of the UDT instance's HighSP memory Tag. This is, in fact, the entire reason why the UDT has that memory Tag in the first place. To do this, we simply need to bind the setpoint to the memory Tag.

1. Click **Amps**.
2. Click **Alarms**.
3. Double-click **New Alarm**.
4. Enter **High Amps Alarm** in the Name field.
5. Select **High** from the Priority list.
6. Select **Above Setpoint** from the Mode list.
7. Click  on the Setpoint field.



8. Click **Tag**.
9. Click **UDT**.
10. Click **SpeedSP**.
11. Click **Apply**.



12. Click **OK**.

Alarms will now be set on the HOA and Amps Tags in each instance of the MotorUDT. You can verify this by looking at the instances in the Tag Browser.

The Alarm Status table will now automatically populate with many more alarms, as the values of the Amps Tags are constantly fluctuating.

Alarm Status Table

Let's take a closer look at some of the built-in features of the Alarm Status Table.

You will notice that there are a set of buttons along the bottom of the table. You will need to be in Preview mode to interact with the table.

Acknowledge

If you select an alarm on the table and click the Acknowledge button, the alarm will transition into the Acknowledged state. You will see the background color of the row change, and its position on the table may change as well. Notice that alarms can still be active even when they have been acknowledged.

Shelve

You can think of shelving an alarm like using the Snooze feature on an alarm on your phone. Shelved alarms are removed from the status table and can be ignored for a pre-set amount of time. Once the time expires, the shelved alarm will display as normal on the table.

Inspect Selected Alarm

If you select an alarm and click the magnifying glass button, you can inspect the alarm to look at details of the alarm, including the exact time and value that the Tag was at when the alarm was triggered.

Alarm Trends

Clicking the button to the right of the magnifying glass will display an alarm trends graph. Note that this is displaying from Tag History data, so it will only work if the Tag is being historized; otherwise, a blank graph will appear.

Manage Shelved Alarms

The final button at the bottom of the table allows operators to view all shelved alarms. They can also unshelve alarms from here.

Sort, Adjust, and Add or Remove Columns

As with other tables in Ignition, the Alarm Status Table includes automatic sorting by clicking on any column header. You can also adjust the width of any column by dragging between column headers. You can right-click on any header to see a list of the available columns; clicking on an unchecked column will add it to the table, while clicking an already-visible column will remove it.

Other Alarm Status Properties

It is also possible to change how the table behaves. For example, all exposed buttons in the table can be shown or hidden using the set of properties that can be found under the Appearance section. You can also bind any of these properties to an expression to limit who can see them.

For example, if you only want those in the Administrator role to be able to shelve alarms, you can do the following:

1. Click the **Alarm Status Table** to select it.
2. Click  on the Show Shelve Button property.

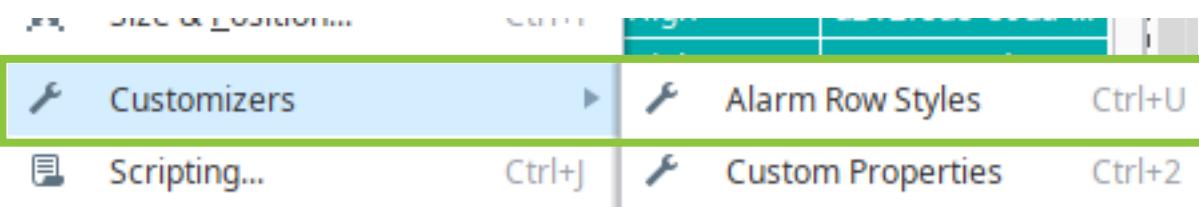


3. Click **Expression**.
4. Enter `hasRole("Administrator")`

Customize Alarm States

You can customize how the various alarm states are displayed as well..

1. Right-click the table.
2. Select **Customizers**.
3. Click **Alarm Row Styles**.



You can see that the alarm rows are based on expression, built around either the state or the priority (or both) of the alarm. You can click on an existing expression and alter any of the formatting, or add your own expression.

Filter the Table

You can set up the table to only display a subset of the alarms using the Display Path Filter property. The property accepts two wildcards: * to represent zero or more characters, and ? to represent a single character.

To set up a filter so that only Motor alarms will display (in our example, to essentially hide the WriteableBoolean1 alarms), you can add a filter that combines those wildcards and the literal text motor.

4. Click  to edit the Display Path filter. You may need to scroll down.
5. Enter ***Motor***
6. Press **Enter**.
7. The table filters to only show alarms that contain the word “motor” in their display path.

Note: Unlike expressions, the display path is not case sensitive.

Dynamic Alarm Filtering

You can also allow your operators to set the alarm filters on the fly by adding a component and a simple binding.

1. Expand the Components Palette.
2. Click  to clear the filter.
3. Enter **Text**.
4. Drag a **Text Field** to the Window.
5. Click  on the Display Path Filter property.
6. Click **Expression**.
7. Enter **"*"** + 
8. Click .
9. Expand **Root Container**.
10. Expand **Text Field**.
11. Click **Text**.
12. Click **OK**.

13. Enter + **"*"**
14. Confirm that your expression looks like this:

```
"*" + {Root Container.Text Field.text} + "*"
```

15. Click **OK**.

You can test this by going into Preview Mode and typing some portion of a display path into the text field.

Custom Display Paths

Operators likely only need to worry about specific alarms on specific machines. Allow for custom filtering of the display path can be helpful, but you can make it even easier for operators to quickly find the alarms they care about by customizing the display path on alarms.

In our example, we are going to change the display path on the UDT Amps alarms so that they read something like “Motor 1 High Amps Alarm,” rather than displaying the entire path of the Tag.

1. Click **UDT Definitions**.
2. Double-click **MotorUDT**.
3. Click **Amps**.
4. Click **Alarms**.
5. Click **High Amps Alarm**.
6. Click  on the Display Path property.
7. Click **Expression**.
8. Enter "Motor " +
9. Click .
10. Click **MotorNum**.
11. Enter + " High Amps Alarm"
12. Ensure your expression looks like this:

```
"Motor " + {MotorNum} + " High Amps Alarm"
```

13. Click **Apply**.
14. Click **OK**.

Any new alarms that are added to the table will show the new display path.

The expression above is one of several possible ways to achieve this result. Because we are working in a UDT, the alarm has access to the name of the instance of the UDT as a parameter, so rather than hard-coding the word "Motor" and appending the **MotorNum** parameter, you could instead use the **InstanceName** parameter, which is available using the same UDT Parameters button we used above. Then, you could add a literal space, and instead of writing "High Amps Alarm" you could instead use the alarm's own **name** property, which you can find by clicking the Alarm Properties button, then navigating to Main, then Name. The resulting expression would be {InstanceName} + " " + {name}. The resulting display path would be identical.

Alarm Associated Data

Sometimes you might want to have more data than just your alarm setpoints and values when an alarm goes off. Alarms can carry any extra data you may want as Associated Data. Associated Data are like custom properties on Vision components; each alarm can have as many Associated Data Properties as you want, and these properties can be bound.

For example, if your Motor Amps goes into its alarm state, you might want to know what the HOA value was when the alarm happened. Instead of having to check Tag history and hope the value was recorded, you can add the HOA value directly to the alarm.

1. Double-click **MotorUDT**.
2. Click **Amps**.
3. Click **Alarms**.
4. Click **Amps High Alarm**.
5. Click .

An Associated Data property is added at the bottom of the list.

6. Double-click **New Data**. You may need to scroll down.
7. Enter **HOA**.
8. Click .
9. Click **Tag**.
10. Click **HOA**.

11. Click **Apply**.
12. Click **OK**.
13. Click .
14. Select a **Motor High Amps** alarm.
15. Click **Inspect selected alarm**.
16. Scroll down on the list of **Details**.

Near the bottom of the list, you will see the HOA property. It will likely be empty; while all alarms will now have the property, only alarms triggered after you add the property will have a value.

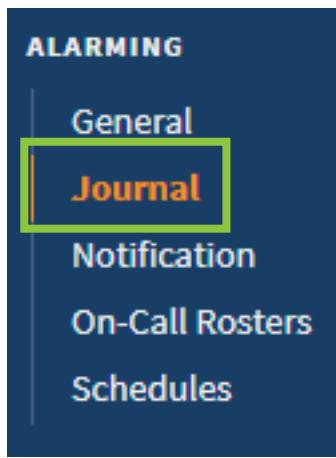
Alarm History

So far, we have been working only with the Alarm Status Table, which is showing us a live list of the alarms our Gateway is storing in memory. However, this system does not keep track of information forever—if we want to store alarm events in the database, we'll need to set up an Alarm Journal.

Create an Alarm Journal

An Alarm Journal allows us to store alarm events in our database, so that they can be retrieved at any time in the future. The Alarm Journal is created in the Gateway web page.

1. Open the **Gateway** web page.
2. If necessary, log in.
3. Click **Config**.
4. Click **Journal** in the Alarming section.



5. Click **Create new Alarm Journal Profile...**

Alarm Journal Storage Options

When setting up an Alarm Journal, you have three storage options.

Database Use an existing database connection within your Gateway. To review how to set up a database connection, see “Connecting to a Database” on page 44.

Remote Connect to another Gateway to use its Alarm Journal.

Internal Use an internal SQLite database to store the information. Note that this differs from the Database option above in that the data if stored internally can only be accessed from within Ignition, while the Database option stores the data in a separate database that can be accessed directly, without Ignition.

Configure the Alarm Journal

1. Click **Database**.

2. Click **Next**.
3. Enter **Journal** in the Name field.

Note: Be sure to capitalize the J.

Main	
Name	Journal
Datasource	DB Events will be stored to this datasource.
Enabled	<input checked="" type="checkbox"/> (checkbox)

4. Click **Create New Alarm Journal Profile**. You may need to scroll down.

Additional Settings

While we are not going to change any other settings, there are a few that are noteworthy.

Events You can set the priority level of alarms you wish to store, as well as determining whether or not to store shelved, enabled, and disabled events.

Store Shelved Events You can choose whether or not alarms that are shelved are still stored in the Journal.

Data Filters The three options here allow you to set up filters on the Alarm Source, Display Path, or Display Path or Source. These filters follow the same patterns as filtering on the Alarm Status table. See “Security” on page 231.

Data Pruning You can have Ignition automatically deleted old data from the database by enabling pruning and then determining the age at which events are deleted. Note that once pruned, the data cannot be recovered.

Viewing the Journal

Once the Journal is set up, Ignition will automatically create two new tables in the database and begin storing the data as soon as an alarm occurs.

Inspect the Database Table

You can see these tables and their data by looking at the database directly.

1. Return to **Designer**.
2. Click **Tools**.
3. Click **Database Query Browser**.
4. Double-click **alarm_events**.
5. Click **Execute**.

The screenshot shows the Database Query Browser interface. In the top-left, there's a query editor with the SQL command: `SELECT * FROM alarm_events`. To the right of the query is a green-bordered 'Execute' button. Below the query editor, the 'Resultset 1' pane displays a table with 9 rows of alarm event data. The columns are: id, eventid, source, displaypath, priority, eventtype, eventflags, and time. The data includes various alarm types like 'High Amps Alarm' and 'Low Amps Alarm' from different sources. On the far right, the 'Schema' pane is open, showing the database structure with 'alarm_events' highlighted. At the bottom of the browser, it says '9 row(s) fetched in 4 ms'.

id	eventid	source	displaypath	priority	eventtype	eventflags	time
1	722763e1-5ad0-46f1-89c8-83...	prov:default:/tag:GenSim/Writ...		2	0	0	2024-08-15 10:30:00
2	722763e1-5ad0-46f1-89c8-83...	prov:default:/tag:GenSim/Writ...		2	1	16	2024-08-15 10:30:00
3	879fa732-17fd-4bf5-ba7e-885...	prov:default:/tag:GenSim/Writ...		2	0	0	2024-08-15 10:30:00
4	7a9ddd14-2399-42e3-9ff7-aa...	prov:default:/tag:Motors/Moto...	Motor 2 High Amps Alarm	3	1	16	2024-08-15 10:30:00
5	71d2f812-b94d-4925-8480-27...	prov:default:/tag:Motors/Moto...		3	2	20	2024-08-15 10:30:00
6	325e9364-1677-4843-9763-c7...	prov:default:/tag:Motors/Moto...	Motor 2 High Amps Alarm	3	0	0	2024-08-15 10:30:00
7	b2fc0d58-669a-45b5-bec4-a5...	prov:default:/tag:Motors/Moto...	Motor 9 High Amps Alarm	3	1	16	2024-08-15 10:30:00
8	e9ba4a69-ba61-48c6-b178-3c...	prov:default:/tag:Motors/Moto...		3	2	20	2024-08-15 10:30:00
9	9207a595-7315-441d-90a0-e9...	prov:default:/tag:Motors/Moto...	Motor 9 High Amps Alarm	3	0	0	2024-08-15 10:30:00

This table shows the alarm events, and includes columns for the source (the Tag that generated the alarm), the display path (if set), the priority, time the alarm occurred, and more.

6. Double-click **alarm_event_data**.

7. Click **Execute**.

This table shows the actual values of the Tags at the moment the alarm was triggered. The id column on this table relates it to the id column in the alarm_events table. If you know SQL, you can write queries to join these tables and see the data more clearly. However, we don't need to do that, as we have a component that is built specifically to display Alarm Journal information.

Add an Alarm Journal Table

1. If necessary, open the **Alarms** window in Vision.
2. Right-click the **Alarm Status Table**.
3. Click **Layout**.
4. Click **Anchor South** to turn it off.
5. Click **OK**.
6. Click **Root Container** in the Project Browser.
7. Drag the **bottom middle control handle** to make the window about twice as tall.
8. Expand the **Components Palette**.
9. Click in the **Filter** box.
10. Enter **Alarm**.
11. Drag an **Alarm Journal Table** to the window.
12. Position and resize the table to fit in the bottom of the window.

<input type="checkbox"/>	8/9/24, 9:12 PM	Motors/Motor 7/Amps/High Amps Alarm			Cleared, Unacknowl...	High	4b482ba4-0fe4...			
<input type="checkbox"/>	8/9/24, 9:20 PM	Motors/Motor 2/Amps/High Amps Alarm			Cleared, Unacknowl...	High	fe8f756c-243e-4...			
<input type="checkbox"/>	8/9/24, 9:19 PM	Motors/Motor 7/Amps/High Amps Alarm			Cleared, Unacknowl...	High	4e99631e-f5a9-...			
<input type="checkbox"/>	8/9/24, 9:05 PM	Motors/Motor 3/Amps/High Amps Alarm			Cleared, Unacknowl...	High	e9b6b9a5-e9c3...			
	Acknowledge	Shelve								
1	Event Time	Event Id	Display Path	Event State	Priority	System Ev...	Ack'ed By	Event Value	Current St...	Label
2	8/9/24, 9:52 PM	af5e459...	Motor 6 High A...	Active	High	False		53.3808	Active, U...	High Am...
3	8/9/24, 9:52 PM	dc1c56d...	Motors/Motor ...	Ack	High	False	Live Eve...		Cleared, ...	High Am...
4	8/9/24, 9:52 PM	2558686...	Motor 5 High A...	Active	High	False		53.3808	Active, U...	High Am...
5	8/9/24, 9:52 PM	8ec0efb...	Motors/Motor ...	Ack	High	False	Live Eve...		Cleared, ...	High Am...
6	8/9/24, 9:52 PM	a45d4b6...	Motor 6 High A...	Clear	High	False		35.7102	Cleared, ...	High Am...
7	8/9/24, 9:52 PM	caef2c2e...	Motor 5 High A...	Clear	High	False		35.7102	Cleared, ...	High Am...
8	8/9/24, 9:51 PM	863cae9...	Motor 9 High A...	Active	High	False		51.7499	Active, U...	High Am...
9	8/9/24, 9:51 PM	744c2ed...	Motors/Motor ...	Ack	High	False	Live Eve...		Cleared, ...	High Am...
10	8/9/24, 9:51 PM	9207a59...	Motor 9 High A...	Clear	High	False		49.7223	Cleared, ...	High Am...
11	8/9/24, 9:51 PM	2ed797b...	Motor 1 High A...	Active	High	False		53.7663	Active, U...	High Am...
12	8/9/24, 9:51 PM	8456c1f...	Motor 1 High A...	Clear	High	False		42.6248	Cleared, ...	High Am...
13	8/9/24, 9:49 PM	9207a59...	Motor 9 High A...	Active	High	False		50.2168	Active, U...	High Am...
14	8/9/24, 9:49 PM	e9ba4a6...	Motors/Motor ...	Ack	High	False	Live Eve...		Cleared, ...	High Am...
15	8/9/24, 9:49 PM	b2fc0d5...	Motor 9 High A...	Clear	High	False		9.3628	Cleared, ...	High Am...
16	20 events									

The table will automatically populate with the Alarm Journal data.

Note: The Alarm Journal Table assumes that the journal is named *Journal* in the Gateway. If the journal has any other name (and it is case-sensitive, so the “J” needs to be capitalized,) then the table will not automatically populate. You can solve this by clicking  on the Journal Name property and typing the name you gave the Journal in the Gateway.

Configure the Alarm Journal Table

By default, the Alarm Journal Table displays the last 8 hours of information. It populates with the latest data when it first renders, meaning whenever the window it is on is opened, but it will not automatically update as new data comes in. You can force updates to the table by changing the Start Date and End Date properties of the table. Let's take a look at two ways that we can do that.

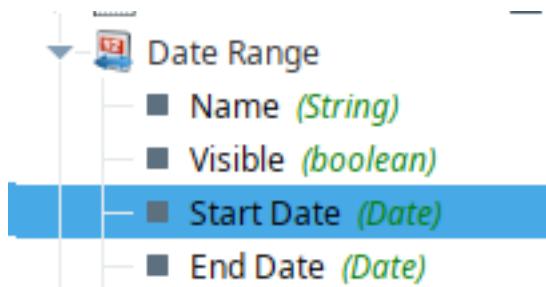
Add a Date Range Component

One option to update the Journal's data is to bind it to a Date Range component. Once bound, changing the date range will cause the table's Start Date and End Date properties to update, which will force the table to get new data.

1. Click the **Alarm Journal Table**.
2. Resize the table to provide about an inch of space above it.
3. Expand the **Components Palette**.
4. Click in the **Filter** box.
5. Enter **Date**.
6. Drag a **Date Range** component to the window.

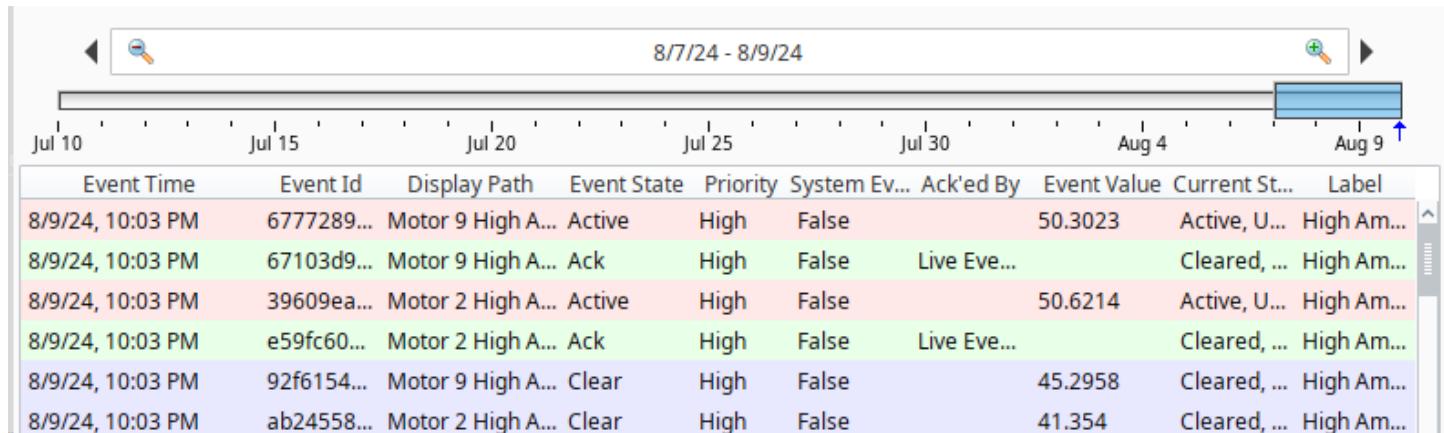
Note: We are using a Date Range component in this example, but any of the date picker components will work.

7. Click the **Alarm Journal Table**.
8. Click  on the table's **Start Date** property.
9. Click **Property**.
10. Expand **Date Range**.
11. Click **Start Date**.



12. Click **OK**.
13. Click  on the table's **End Date** property.
14. Click **Property**.
15. Expand **Date Range**.
16. Click **End Date**.

17. Click **OK**.
18. Click .
19. Drag the **slider** on the Date Range component to change the date.



The screenshot shows a software interface with a toolbar at the top containing icons for search, refresh, and other functions. Below the toolbar is a date range selector labeled "8/7/24 - 8/9/24" with a slider bar. The table below has columns for Event Time, Event Id, Display Path, Event State, Priority, System Ev..., Ack'ed By, Event Value, Current St..., and Label. The data rows show various alarms for different motors with their respective details like priority and event state.

Event Time	Event Id	Display Path	Event State	Priority	System Ev...	Ack'ed By	Event Value	Current St...	Label
8/9/24, 10:03 PM	6777289...	Motor 9 High A...	Active	High	False		50.3023	Active, U...	High Am...
8/9/24, 10:03 PM	67103d9...	Motor 9 High A...	Ack	High	False	Live Eve...		Cleared, ...	High Am...
8/9/24, 10:03 PM	39609ea...	Motor 2 High A...	Active	High	False		50.6214	Active, U...	High Am...
8/9/24, 10:03 PM	e59fc60...	Motor 2 High A...	Ack	High	False	Live Eve...		Cleared, ...	High Am...
8/9/24, 10:03 PM	92f6154...	Motor 9 High A...	Clear	High	False		45.2958	Cleared, ...	High Am...
8/9/24, 10:03 PM	ab24558...	Motor 2 High A...	Clear	High	False		41.354	Cleared, ...	High Am...

The table updates with new data.

Update with Expressions

You can also force the table to update by using expressions on the Start Date and End Date properties. Thankfully, we have some built-in functions to make this simple.

To show this, we're going to add a second Alarm Journal Table to the window, so that we can see both options at work.

1. Click **Root Container** in the Project Browser.
2. Drag the **bottom middle control handle** to make the window taller.
3. Expand the **Components Palette**.
4. Click  to clear the Filter box.
5. Enter **Alarm**.
6. Drag an **Alarm Journal Table** to the window.
7. Position and resize the table to fit in the bottom of the window.

We will start by building the Start Date expression. For this example, we want to get the last hour of data, and have the table update every 30 seconds.

There are no functions in the Expression language to subtract time. However, there are a set of functions to add time, all of which can take negative values, which has the effect of subtracting time.

8. Click ► on the table's **Start Date** property.
9. Click **Expression**.
10. Click Σ .
11. Select **Date and Time**.
12. Select **add<field>**.
13. Click **addHours**.

The screenshot shows the SAP Fiori Launchpad interface. A context menu is open over a table column labeled "Start Date". The menu has several sections:

- Date and Time** (highlighted in blue)
- Advanced**
- Aggregates**
- Alarming**
- Colors**
- JSON**
- Logic**
- Math**
- Strings**
- Translation**
- Type Casting**

Under the "Date and Time" section, the "addHours" function is highlighted with a green box. The "addHours" function is described as "returns Date". To the right of the menu, a list of date-related functions is shown, each with its name, description, and return type. The "addHours" function is the second item in this list.

Function	Description	Return Type
<code>addDays(date, number)</code>	returns Date	
<code>addHours(date, number)</code>	returns Date	
<code>addMillis(date, number)</code>	returns Date	
<code>addMinutes(date, number)</code>	returns Date	
<code>addMonths(date, number)</code>	returns Date	
<code>addSeconds(date, number)</code>	returns Date	
<code>addWeeks(date, number)</code>	returns Date	
<code>addYears(date, number)</code>	returns Date	
<code>dateArithmetic(date, number, field)</code>	returns Date	
<code>dateDiff(date, date, field)</code>	returns Double	
<code>dateExtract(date, field)</code>	returns Integer	
<code>dateFormat(date, pattern)</code>	returns String	
<code>dateIsAfter(date, date)</code>	returns Boolean	
<code>dateIsBefore(date, date)</code>	returns Boolean	
<code>dateIsBetween(date, date, date)</code>	returns Boolean	
<code>dateIsDaylight([date])</code>	returns Boolean	
<code>fromMillis(long)</code>	returns Date	
<code>get<field></code>		
<code>getDate(int, int, int)</code>	returns Date	
<code>getTimezone()</code>	returns String	
<code>getTimezoneOffset(date)</code>	returns Double	
<code>getTimezoneRawOffset()</code>	returns Double	
<code>midnight(date)</code>	returns Date	
<code>now([pollRate])</code>	returns Date	
<code>setTime(date, int, int, int)</code>	returns Date	
<code>timeBetween(date, date, date)</code>	returns Boolean	
<code>toMillis(date)</code>	returns Long	

At the bottom right, there are settings for the table:

- Show Table Header: checked (true)
- Selection Color: 0,0,0 (represented by a black square)
- Selection Thickness
- Table Background: 255,255,255
- Row Height
- Row Styles: Dataset [6R x 8C]

The `addHours` function needs two parameters: a time to start from, and a number of hours to add (or subtract) from that time. For this expression, we want to start from the current time, which we can get thanks to the `now()` function, and add negative one hour to that.

14. Enter `now()`, `-1` inside the parentheses to complete the expression.

15. Check that your expression looks like this:

```
addHours(now(), -1)
```

We're almost there, but not quite yet. By default, `now()` updates (or polls) to get the current time every second. We do not want this; instead, we want to update the time every 30 seconds. Fortunately, `now()` can take a parameter to set the polling frequency, in milliseconds.

16. Click between the parentheses after `now`.

17. Enter `30000`.

18. Check your final expression:

```
addHours(now(30000), -1)
```

19. Click **OK**.

Now that we have the Start Date expression set, we need to add a simpler expression to End Date. There, we only need to tell it to start at the current moment, updated every 30 seconds.

20. Click  on the table's **End Date** property.

21. Click **Expression**.

22. Enter `now(30000)`.

23. Check your expression:

```
now(30000)
```

24. Click **OK**.

You should see the table update with new data. Wait 30 seconds, and it will update again.

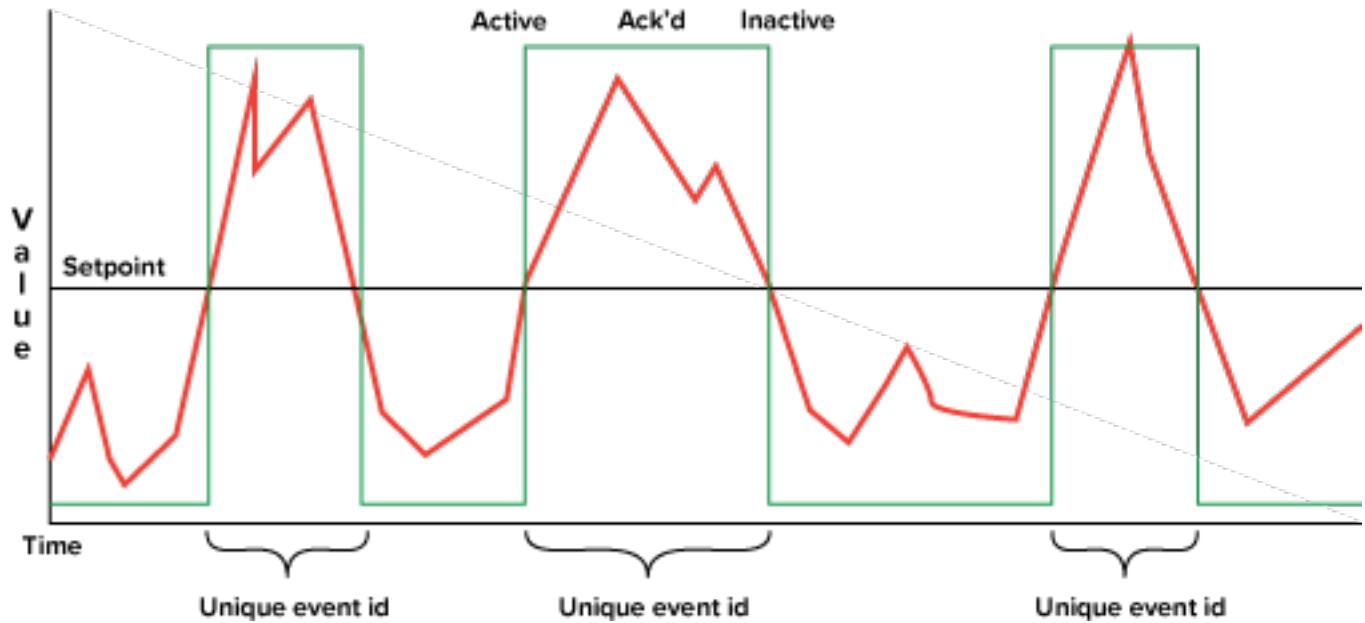
Alarm Event Ids

Before we explore the Alarm Journal Table further, we need to discuss what is meant by an “alarm event,” and understand the alarm event life cycle.

Every time an alarm triggers—in other words, the alarm condition is true—an alarm event is triggered and the alarm becomes active. At some future point, the alarm condition will become false, and the alarm becomes inactive. In between those two points, the alarm can be acknowledged.

The source of an alarm is the Tag, and a single Tag can and will generate multiple alarms. All of those alarms would therefore be from the same source, but each time an alarm becomes active, it generates a new event, which has a unique event id.

The following diagram illustrates this. In the diagram, the orange line represents a Tag's value over time. Each time the value exceeds the setpoint (represented by the straight horizontal line on the chart), an alarm becomes active. Each time the value drops below that setpoint, the alarm becomes inactive. Each of these active/inactive instances generates a unique alarm id, even though all alarms are from the same source.



Alarm Journal Table Features

Now that you have an understanding of how alarm ids are generated, let's look at the features of the alarm table. Note that you will need to be in Preview mode to use the buttons on the table.

Event Counter The bottom left corner of the table shows the number of events currently displaying in the table.

Focus The small bullseye button in the bottom right of the table allows you to focus, or filter, the table by alarm source or by event id. You will need to have an alarm selected for this button to be active.

Inspect Selected Alarm This allows you to see details on the alarm. The button functions exactly as the same button on the Alarm Status Table.

Filter The Journal Table provides built-in functionality for creating filters.

Alarm Notification

Alarm notification is the act of sending a message to a group of people when an alarm becomes active or clear. In Ignition, this functionality is enabled by having the Alarm Notification Module installed. You can also add the Voice and SMS notification modules for additional ways to notify users.

Alarm Notification Process

In order to set up alarm notifications, you need to complete five steps in order. The first three are completed on the Gateway, and the final two in Designer. The steps are:

1. Add user contact information
2. Create a roster, or a list of users to be notified
3. Create an Alarm Notification Profile
4. Create an Alarm Pipeline
5. Connect an alarm to a pipeline

The remainder of this chapter will go over these steps, one at a time.

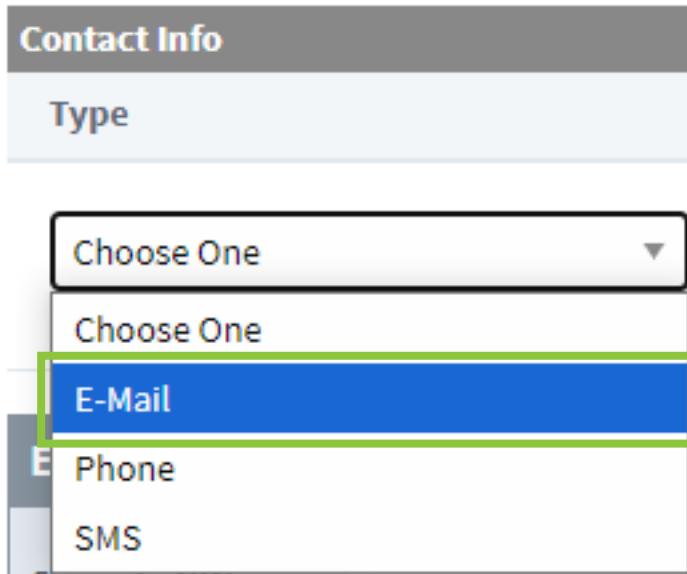
Add User Contact Information

In chapter 13, Security, we created a set of roles and users. Now, we will add contact information to those user accounts.

1. Return to the **Gateway web page**.
2. If necessary, **log in**.
3. Click **Config**.
4. Click **Users/Roles** under the Security heading.
5. Click **More** to the right of default.
6. Click **manage users**.
7. Click **edit** to the right of admin.

Username	Name	Roles	Contact Info	Schedule	
admin		Administrator		Always	Edit
-					Delete

8. Scroll down to the **Contact Information** section.
9. Click **Add Contact Info**.
10. Select **Email** from the list.



11. Enter **admin@company.com**.

Note: The exact address is unimportant for class, but it does need to be formatted as a valid address. For our purposes, we will set all emails to the same format: username@company.com.

12. Click **Save**.



13. Click **Save User**.

14. Repeat steps 7-13 for **each of the other users**, except Guest.

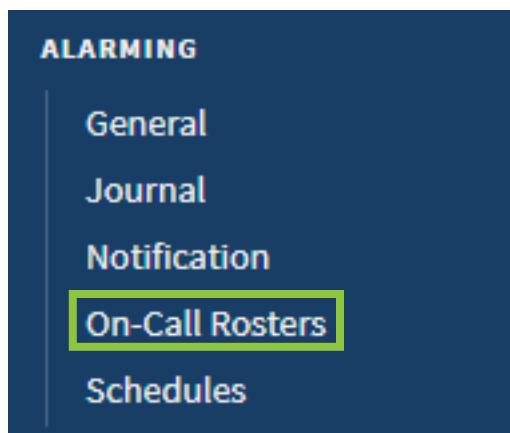
Username	Name	Roles	Contact Info
admin		Administrator	email: admin@acme.com
guest			
manager		Manager	email: manager@acme.com
oper		Operator	email: oper@acme.com
supervisor		Manager, Administrator	email: supervisor@acme.com

Create a Roster

Once we have the contact information entered, you need to group users into rosters. Just as security relies on roles, rather than individual users, alarm notification relies on rosters.

For class, we will create two rosters. The first will be for our managers, which will include our admin, supervisor, and manager roles. The second will be for our operators, which will include our manager and our operator. Note that the manager user will be included in both rosters.

1. Click **On-Call Rosters** under the Alarming heading.



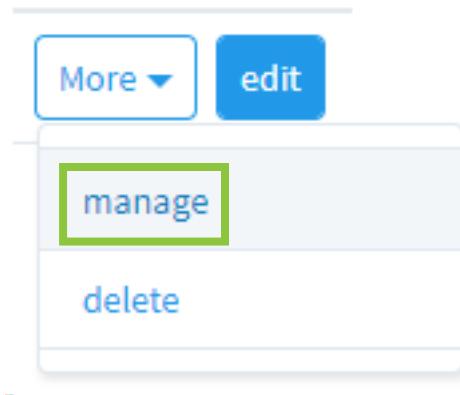
2. Click **Create new On-Call Roster**.
3. Enter **Managers**.
4. Click **Create New On-Call Roster**.

Properties

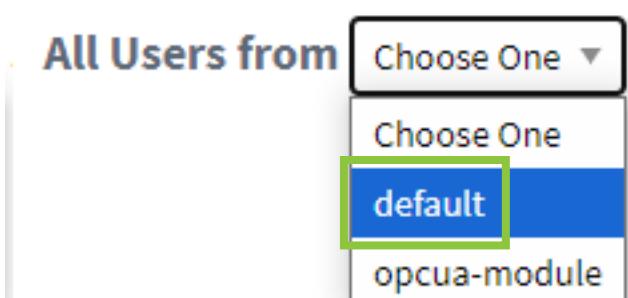
Name	Managers
Description	

Create New On-Call Roster

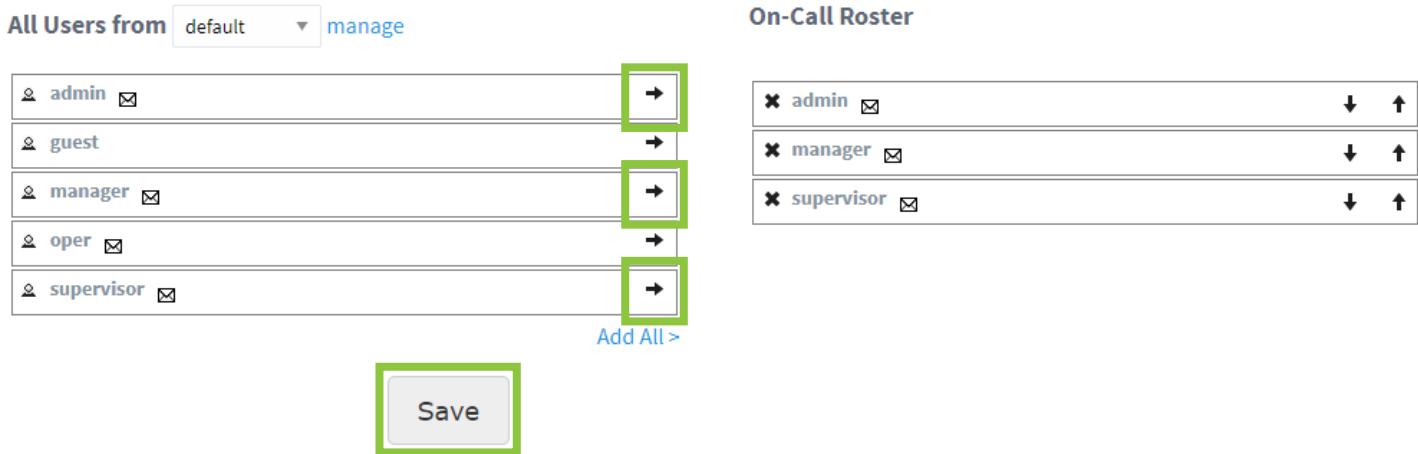
5. Click **More**.
6. Click **Manage**.



7. Select **default** from the **All Users from** list.



8. Click → next to **admin**.
9. Click → next to **supervisor**.
10. Click → next to **manager**.
11. Click **Save**.



12. Click **Create new On-Call Roster**.
13. Enter **Operators**.
14. Click **Create New On-Call Roster**.
15. Click **More**.
16. Click **Manage**.
17. Select **default** from the All Users from list.
18. Click → next to **manager**.
19. Click → next to **oper**.
20. Click **Save**.

Create an Alarm Notification Profile

The third and final step to be completed on the Gateway is to create an Alarm Notification Profile, which defines a delivery channel through which an alarm notification message may be sent to a user.

The Alarm Notification Module allows you to set up a profile using one of the following options:

Email Notification This is what we will use, so it will be discussed in detail below.

Remote Gateway Notification This allows this Gateway to essentially forward the alarm notification to another Gateway's alarm notification system.

Simple One-Way Email Notification A legacy feature that sends an email whenever an alarm within a specified range becomes active. This does not require the creation of alarm pipelines, but lacks any of the configuration and potential sophistication of the Email Notification option.

SMS Notification Rather than using an email server, you can setup and configure an SMS server to send notifications. Note that you would need to get the SMS Notification Module and add phone numbers, either in place of or in addition to, email addresses, to use this option.

VOIP Voice Notification Using a VOIP server, the Gateway can notify users via phone call. This requires the Phone Notification Module and phone numbers in the user contacts.

Email Notification

We want to go ahead and set up an email notification profile.

1. Return to the **Gateway web page**.
2. If necessary, **log in**.
3. Click **Config**.
4. Click **Notification** under the Alarming heading.
5. Click **Create new Alarm Notification Profile**.

Name	Description
No Alarm Notification Profiles	
→ Create new Alarm Notification Profile...	
→ Test Pipelines and Notification Profiles...	

6. Select **Email Notification**.

7. Click **Next**.

Email Notification
Send alarm notifications via email.

Remote Gateway Notification
Enables alarm notification using notification profiles of a remote gateway. Will also expose the pipelines of the remote gateway directly to tags.

Simple One-way Email Notification
Send alarm notifications via email to users in a user source. Listens for active alarms within a given priority range.

SMS Notification
Send alarm notifications via SMS using your Airlink device.

VOIP Voice Notification
Telephone notification using VOIP, compatible with most SIP based telephony systems.

Next >

8. Enter **Email Notification** in the Name field.

In order to test email notifications, we are going to use a free, open-source tool called PaperCut SMTP. This tool, developed by Changemaker Studios and available at <https://github.com/ChangemakerStudios/Papercut-SMTP>, simulates a simple SMTP server, which is what it used to send email. Every email you have ever received has been sent by an SMTP server. PaperCut also includes a simple email interface that we can use to view the emails being sent out.

Because PaperCut is not a real email server, it does not have security configured, so the setup is very easy. In a true production environment, you would need to discuss with your IT department the setup and configuration needed to send emails via your company's system.

9. Enter **localhost** in the Hostname field.

Main	
Name	<input type="text" value="Email Notification"/>
Description	<input type="text"/>
Enabled	<input checked="" type="checkbox"/> (default: true)

Email Settings	
Use Email Profile?	<input type="checkbox"/> If selected, this notification profile will use one of the gateway defined email profiles. Otherwise, it will use the settings defined here. (default: false)
Email Profile	<input type="button" value="Choose One"/> <small>If Use Email Profile is selected, alarm notifications will be emailed using this profile.</small>
Hostname	<input type="text" value="localhost"/>
	<small>Hostname of the SMTP server to send email through. Only used when Use SMTP Profile is false.</small>
Port	<input type="text" value="25"/>
	<small>Port SMTP service is running on. Only used when Use SMTP Profile is false. (default: 25)</small>
Enable SSL/TLS	<input type="checkbox"/> Connect using SSL/TLS. Only used when Use SMTP Profile is false. (default: false)
Username	<input type="text"/>
	<small>Only used when Use SMTP Profile is false.</small>

10. Scroll down.

Two-way Notification

You have the option of enabling two-way settings in the Email Notification profile. Doing so will cause the outgoing email to include a link that, when clicked on by the operator, allows them to acknowledge the alarm.

11. Check **Two-way Enabled**.

12. Enter **localhost:8088** in the Gateway field.

Note: You must not include http:// when entering the Gateway address.

The screenshot shows a configuration interface for 'Two-way Settings'. The first section, 'Two-way Enabled', has a checked checkbox and a note '(default: false)'. The second section, 'Gateway', contains a text input field with the value 'localhost:8088' highlighted with a green border. Below the input field is a note: 'Address and port this gateway is reachable at. Will be used in notifications'.

13. Scroll down.
14. Click **Create New Alarm Notification** Profile.

Alarm Notification Pipelines

Now that we have configured our user contact information, rosters and alarm notification profiles, we can create a pipeline to bring everything together.

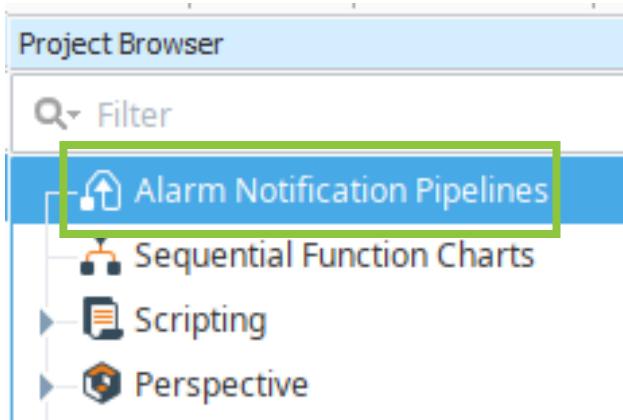
Pipelines are a graphical mechanism for building up logic, providing an easy-to-use drag-and-drop interface for creating complex notification scenarios. Using pipelines, it is possible to accomplish many tasks, such as alarm escalation, notification consolidation, and conditional dispatch.

Because Alarm Notification Pipelines are their own module, they are created in the pipeline workspace. The workspace is effectively a blank canvas, upon which you arrange and connect various pipeline blocks.

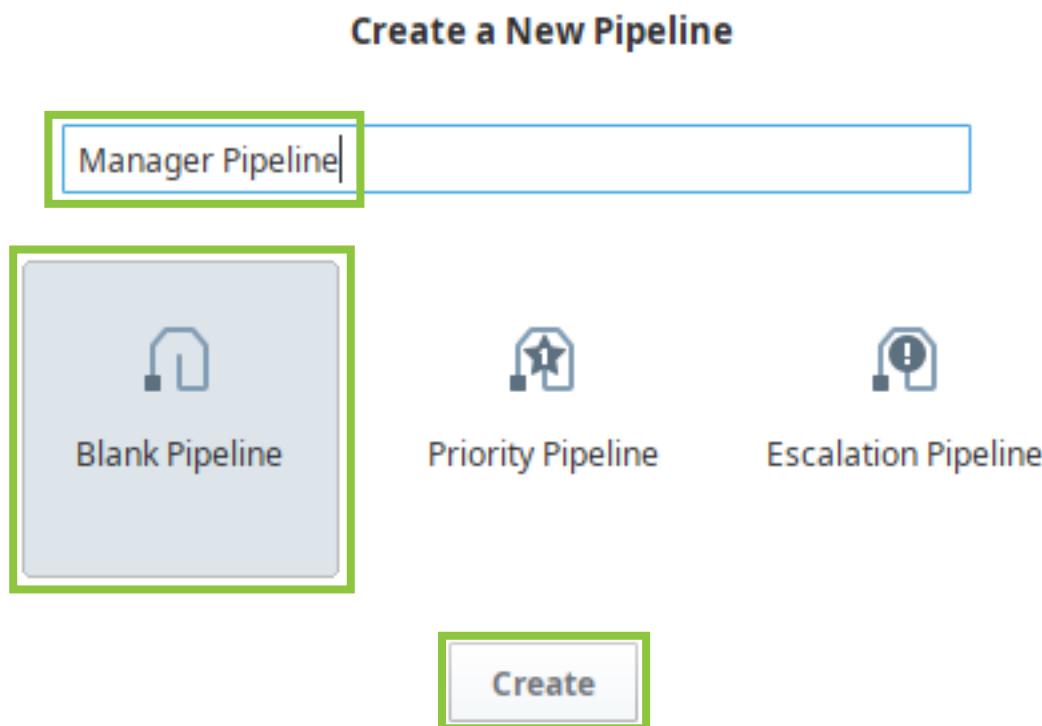
Create a New Pipeline

Let's create a new pipeline. It will use the settings we created in the Gateway to send email notifications to our Managers roster.

1. Switch to **Designer**.
2. Click **Alarm Notification Pipelines**.



3. Enter **Manager Pipeline** in the Name of the pipeline field.
4. Click **Blank Pipeline**.
5. Click **Create**.



Pipeline Blocks

The available blocks are:

Notification Delivers a notification through the selected Notification Profile.

Delay Blocks the alarm event for the specified amount of time.

Splitter Forwards a single event concurrently to multiple other blocks.

Switch Evaluates a non-Boolean expression and forwards to an output based on the result.

Expression Evaluates a Boolean expression and forwards the event to that output.

Set Property Evaluates an expression and sets the result as a runtime property on the alarm event.

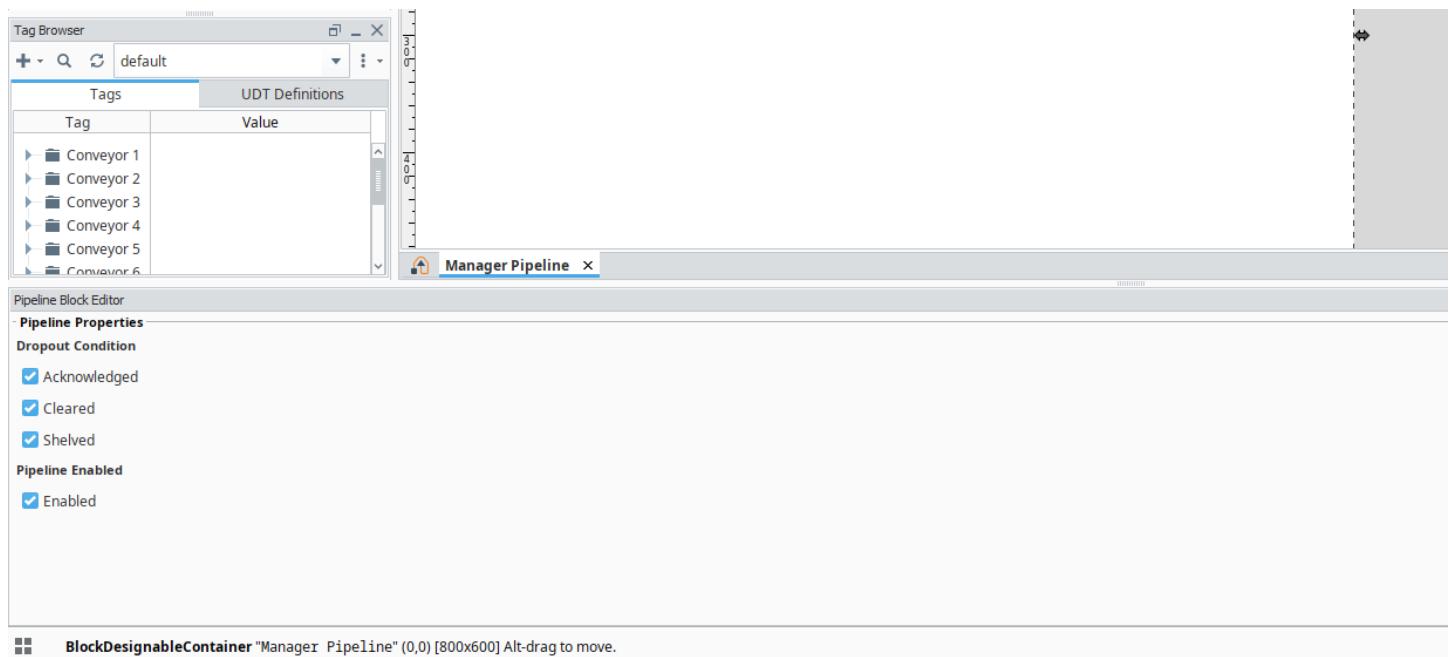
Jump Forwards the alarm event to a different pipeline.

Script Executes a task outside the pipeline.

Pipeline Block Editor

The Pipeline Block Editor serves as a property editor for the Pipelines module. There are not many properties available, but they are almost all important. We want to see these properties more easily, so let's move this panel from the lower left corner to instead stretch across the bottom of the window.

1. Press and hold your mouse over the **Pipeline Block Editor** title.
2. Drag the panel towards the middle of the screen, then down to the bottom.
3. Release your mouse.

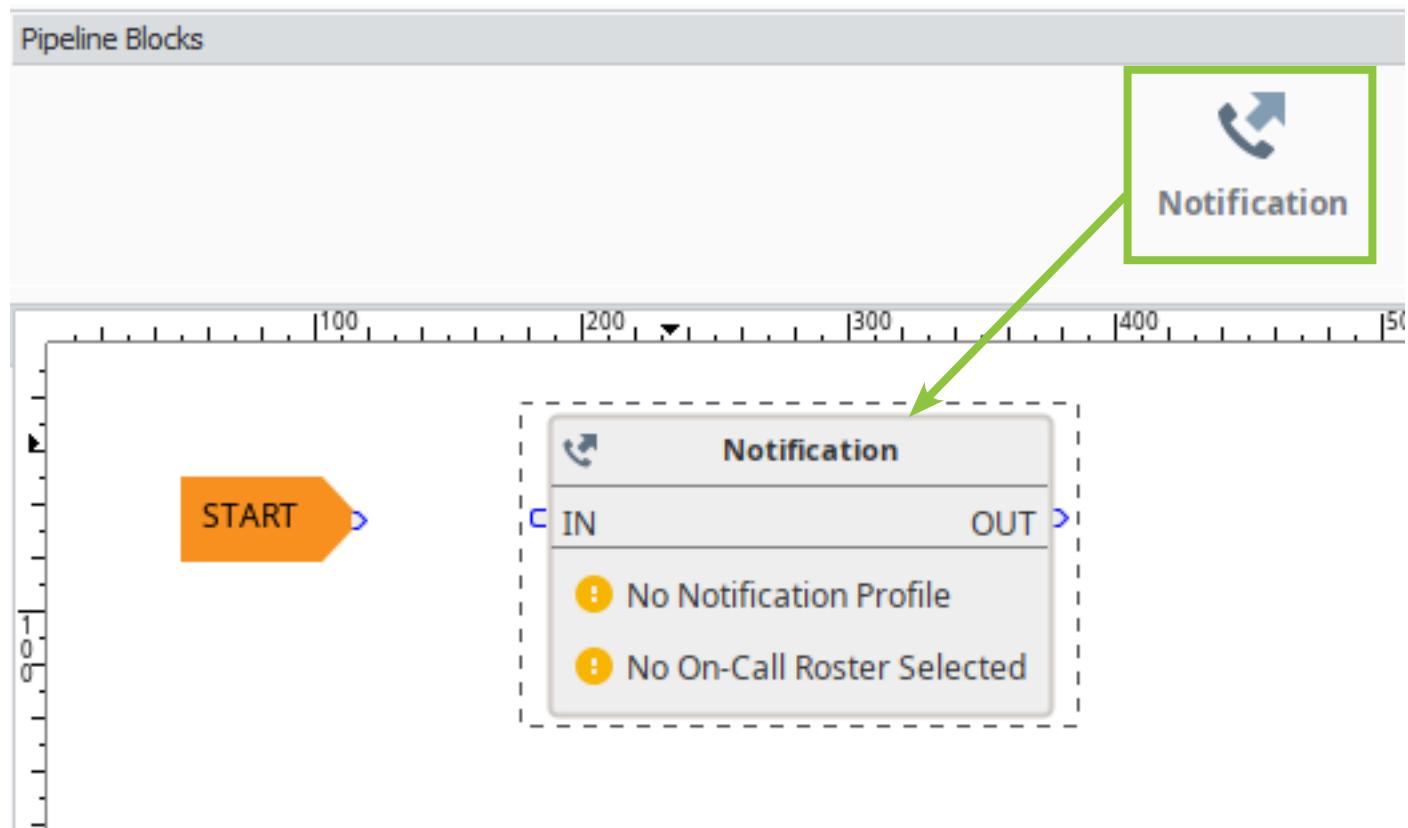


Add Blocks

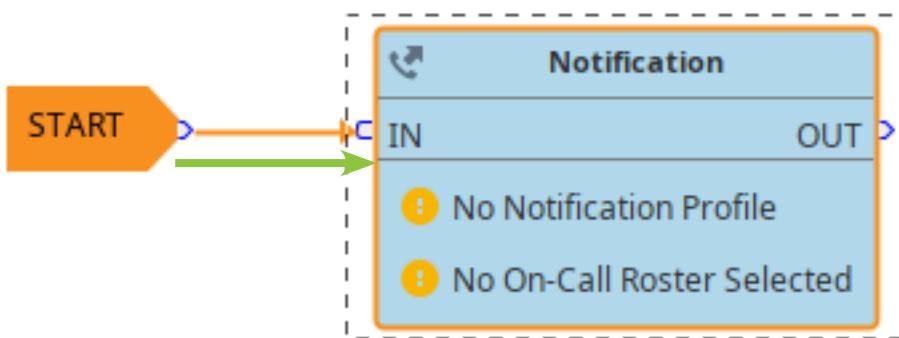
To create the pipeline, we need to add blocks to the main window, then connect them.

All pipelines begin with a START block.

1. Drag a **Notification** block to the window.



2. Press and hold your mouse over the **output pin** on START.
3. Drag to the **input pin** on Notification.

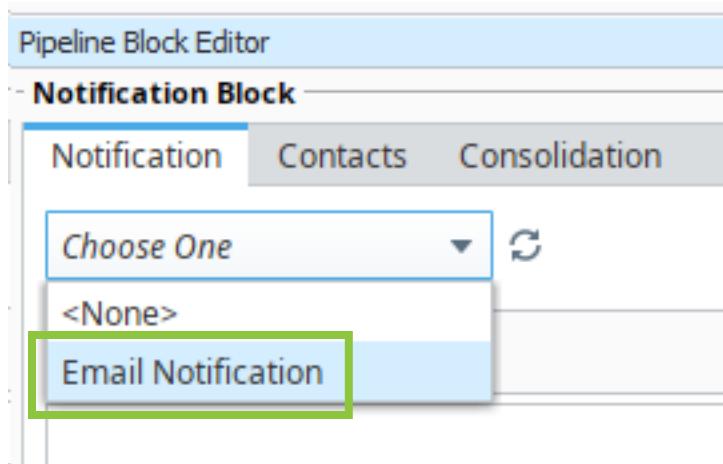


Note: If you find yourself moving the entire Start block, it's likely that you clicked on it and selected it. Click in a blank area of the pipeline editor to deselect, then click-and-drag from the output pin to the input pin.

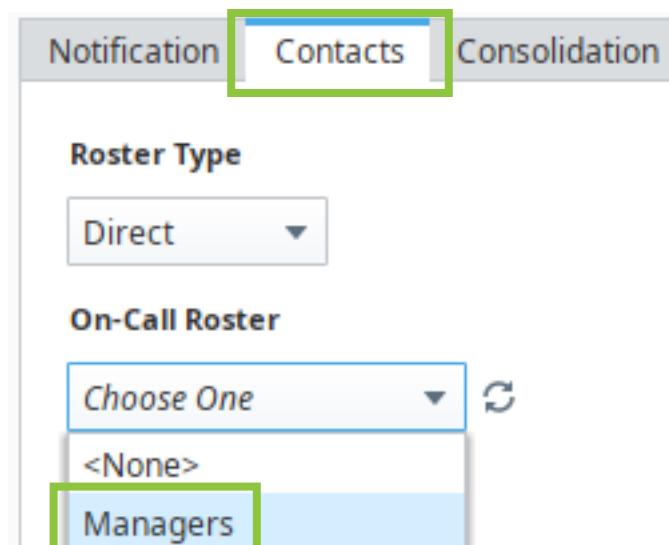
Block Settings

Now that we have added a Notification block, we need to configure its settings. The block has a list of settings needed to fully configure it. For a Notification block, you need to select an Alarm Notification Profile and an On-call Roster.

1. Click the **Notification** block.
2. Select **Email Notification** from the Choose One list in the Pipeline Block editor.



3. Click **Contacts**.
4. Select **Managers** from the On-call Rosters list.

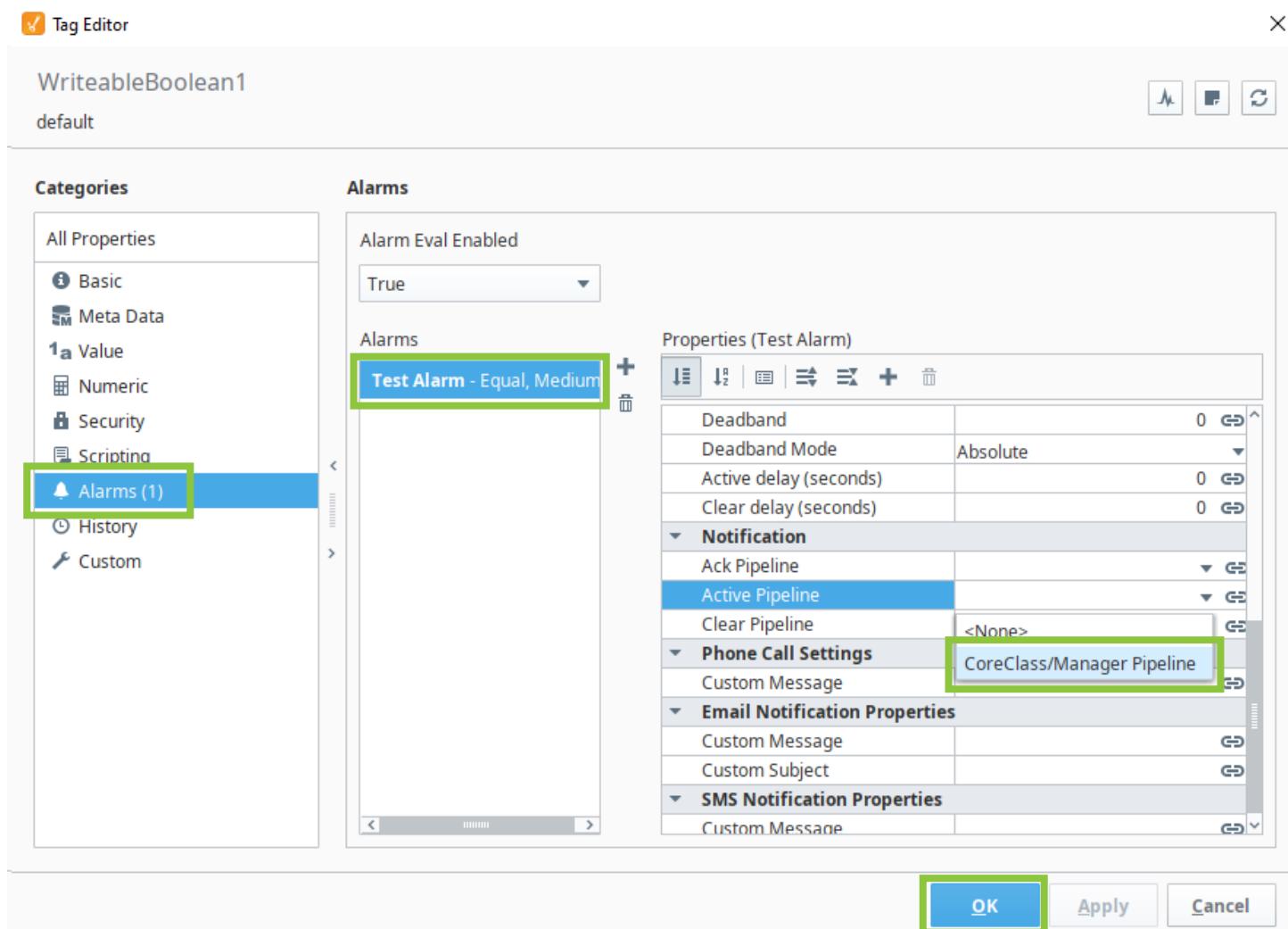


5. Click **File**.
6. Click **Save All**.

Attach an Alarm to a Pipeline

Now that we have the pipeline created, there's only one more step: attaching an alarm to a pipeline.

1. Expand **GenSim** in the Tag Browser.
2. Expand **Writable**.
3. Double-click **WritableBoolean1**.
4. Click **Alarms**.
5. Click **Test Alarm**.
6. Scroll to the **Notification** settings.
7. Select **CoreClass/Manager Pipeline** from the list for Active Pipeline.
8. Click **OK**.



9. Click **File**.

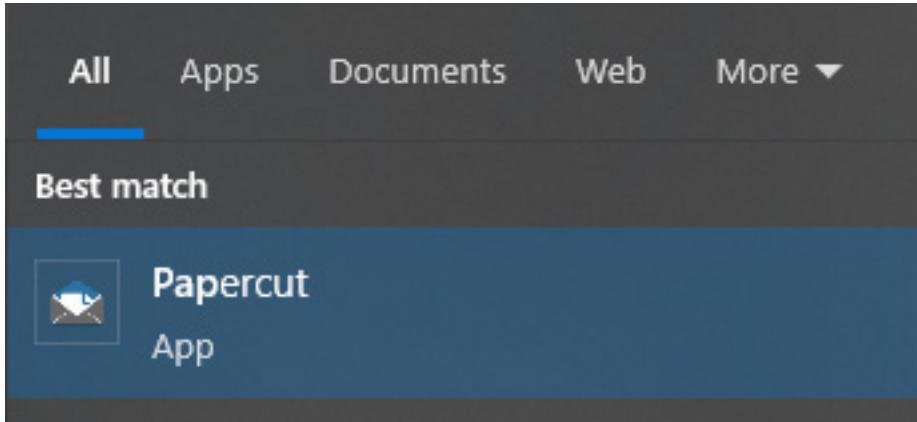
10. Click **Save All**.

Testing the Pipeline

Now that we have completed the five steps to set up the pipeline, we can test it.

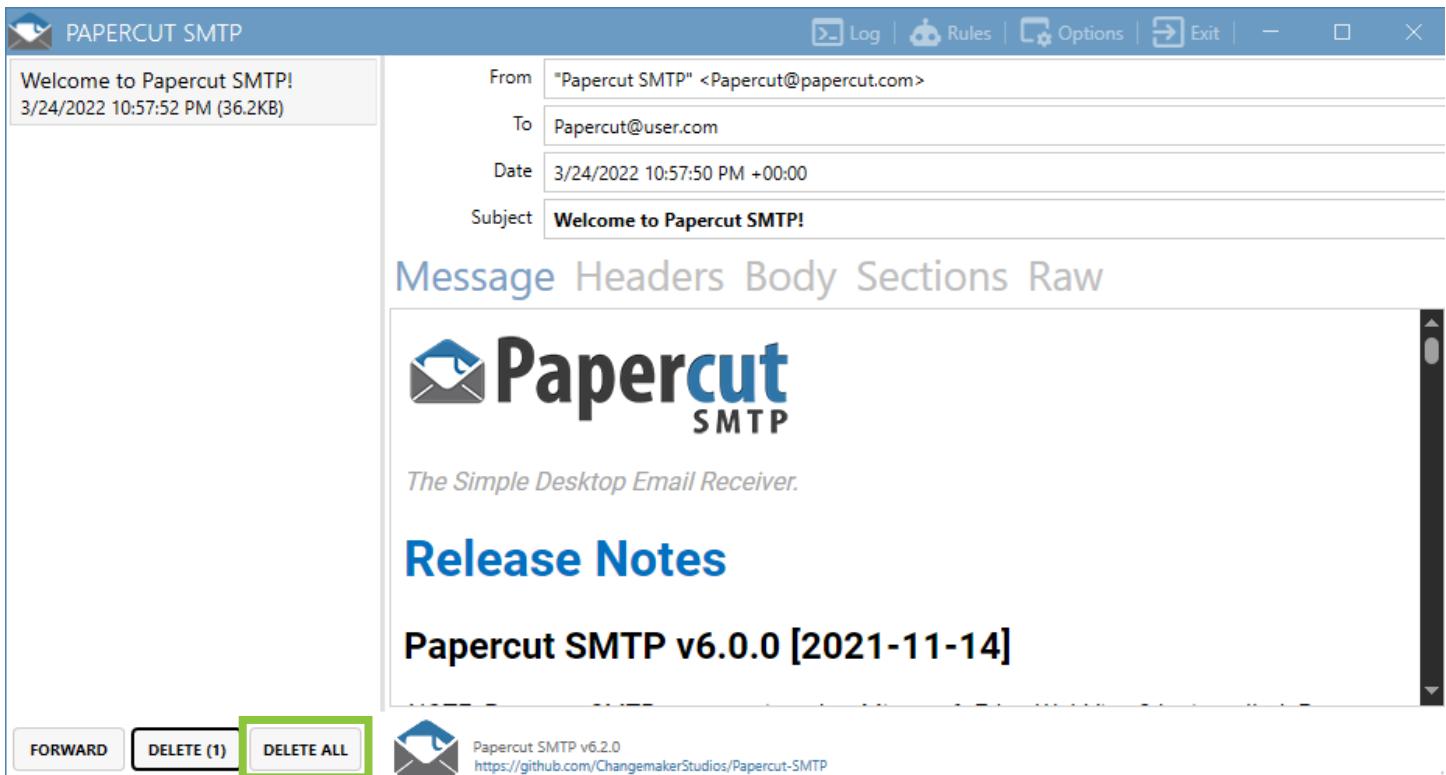
First, we will need to open PaperCut. Then, we will trigger the alarm and view the resulting email.

1. Click the **Windows Start** button.
2. Enter **PaperCut**.
3. Press **Enter**.



Papercut will launch, showing a welcome email. We want to delete this and any other emails currently in Papercut (if any) so that we can focus just on the messages being sent by our system.

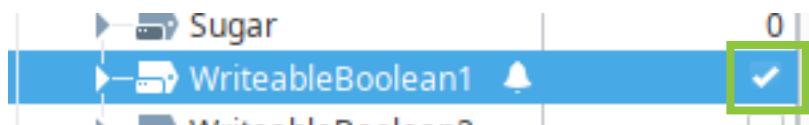
4. Click **Delete All**.
5. Click **Proceed**.



Activate the Alarm

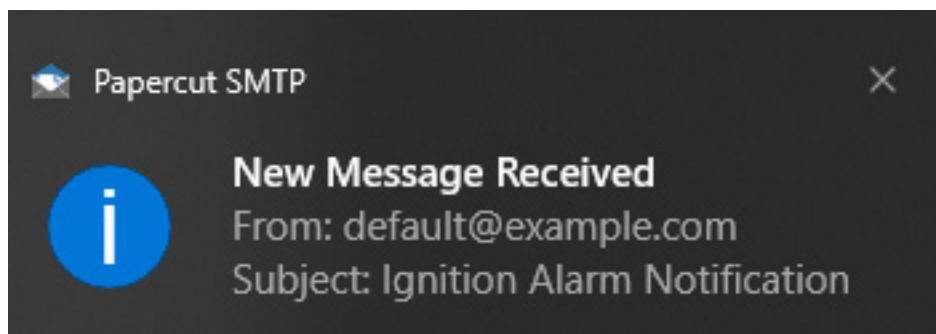
We want to activate an alarm on our WriteableBoolean1 Tag to get the notification email to send.

1. Return to **Designer**.
2. Enable **WriteableBoolean1** in the Tag Browser. If it was already enabled, disable it, then enable it again.



Test the Acknowledgement Link

You should immediately get notifications in Windows that you have new email. We can now test to make sure the two-way settings work.



1. Switch back to **PaperCut**.

You should have three emails, as the roster includes three users (admin, supervisor, and manager.)

Make note of the To: and From: addresses, as well as the body of the email.

Ignition Alarm Notification 8/30/2024 10:19:17 AM (418B)	From: default@example.com
Ignition Alarm Notification 8/30/2024 10:19:17 AM (422B)	To: supervisor@acme.com
Ignition Alarm Notification 8/30/2024 10:19:17 AM (424B)	Date: 8/30/2024 5:19:17 PM +00:00
	Subject: Ignition Alarm Notification

Message Headers Body Sections Raw

```
At 05:19:17, alarm "Test Alarm" at "" transitioned to Active.
http://localhost:8088/web/ack/NvCmgmCx-xVSw-Ttenzh63vtW0GN1yycewD1fTGwfHK1
```

2. Click the **link** in the email.

A web page will open. Take a look at the URL: you will see that it is being sent from your Gateway.

3. Enter a **note**.
4. Click **Acknowledge**.

The screenshot shows a web browser window with the Ignition logo at the top. The address bar displays the URL `localhost:8088/web/ack/NwCmgmCx-xVSw-Ttenzh63vtWO6NlyycewD1fTGWfHKI?0`. The main content area is titled "Ignition Alarm Acknowledgement". It lists a single alarm entry:

Alarm Name	Active Time
Test Alarm @ GenSim/Writeable/WriteableBoolean 1/Test Alarm	5:19 PM

Below the alarm entry is a "Notes:" section containing the text "Alarm acknowledged". At the bottom right of the page is a blue button labeled "Acknowledge".

5. Return to **Designer**.
6. Click **Vision** in the Project Browser.
7. Expand **Windows**.
8. Expand **Main Windows**.
9. Double-click **Alarms**.
10. Switch to **Preview** mode.
11. Click the **acknowledged alarm**.
12. Click **Inspect Selected Alarm**.

13. Scroll to the bottom of the alarm properties.

Active Time	Display Path	Current Sta...	Priority	Event Id
8/30/24, 5:19 PM	GenSim/Writeable/WriteableBoolean1/Test Alarm	Active, Ac...	Medium	ee50bfe1...
8/29/24, 11:34 PM	Motor / High Amps Alarm	Cleared, ...	High	ac99ceb1...
8/29/24, 11:36 PM	Motor 2 High Amps Alarm	Cleared, ...	High	6262e019...
8/30/24, 2:19 PM	Motor 7 High Amps Alarm	Cleared, ...	High	55e24d89...
8/30/24, 2:19 PM	Motor 4 High Amps Alarm	Cleared, ...	High	b3d68e68...
8/30/24, 2:19 PM	Motor 2 High Amps Alarm	Cleared, ...	High	e007736d...
8/30/24, 2:47 PM	Motor 9 High Amps Alarm	Cleared, ...	High	dcf5032b...
8/30/24, 2:49 PM	Motor 1 High Amps Alarm	Cleared, ...	High	0d8d02da...
8/30/24, 3:07 PM	Motor 3 High Amps Alarm	Cleared, ...	High	cff0a458-5...

Details	Notes
name	Test Alarm
Event Time	8/30/24, 5:19 PM
priority	Medium
On Ack	
setpointA	1
Ack'd By	usr-prov:default:/usr:supervisor
activePipeline	CoreClass/Manager Pipeline
name	Test Alarm
Event Time	8/30/24, 5:29 PM
Ack Notes	Alarm acknowledged
priority	Medium

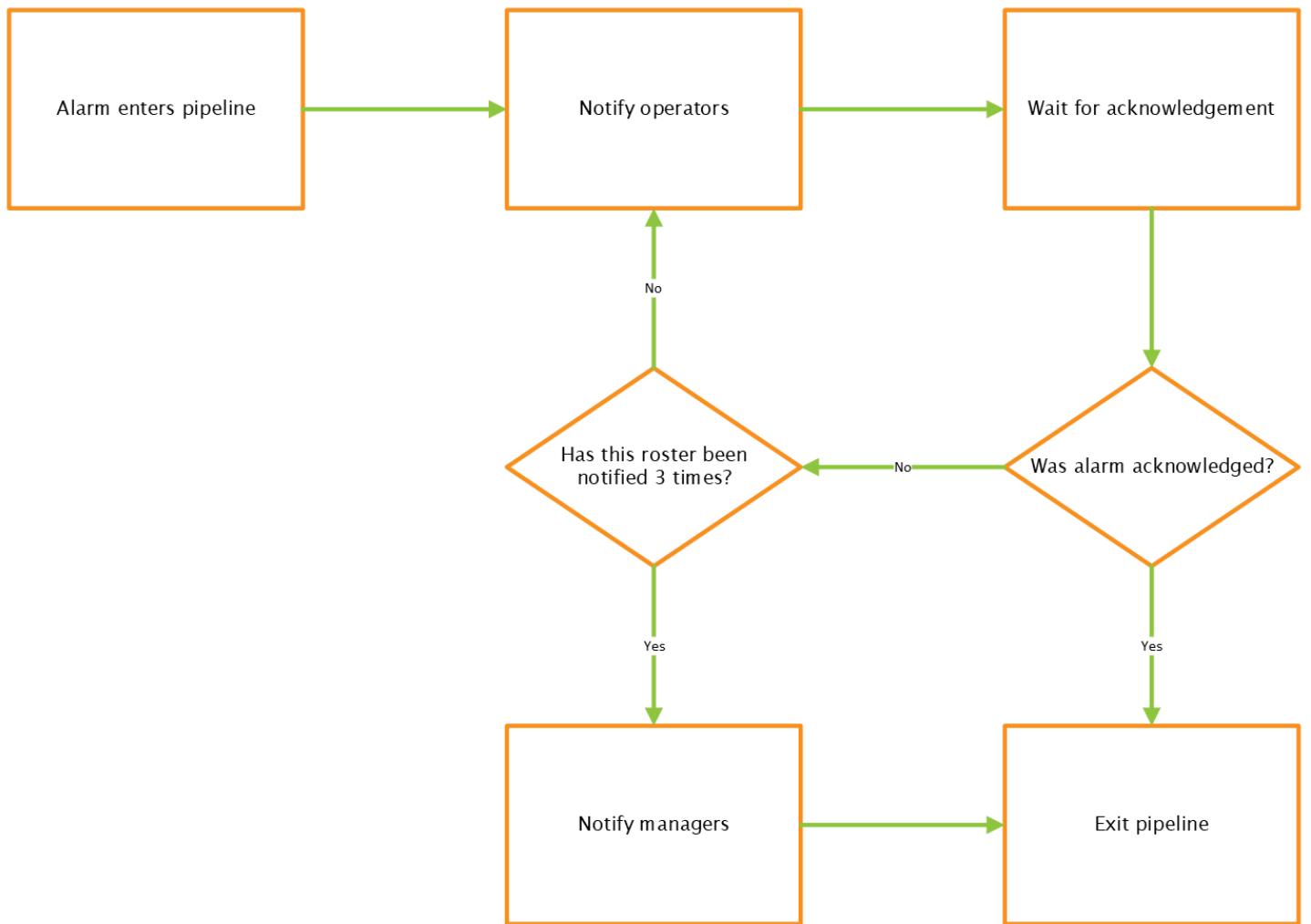
You will see that the properties show that the alarm was acknowledged and by whom, including the note you entered.

Escalation Pipeline

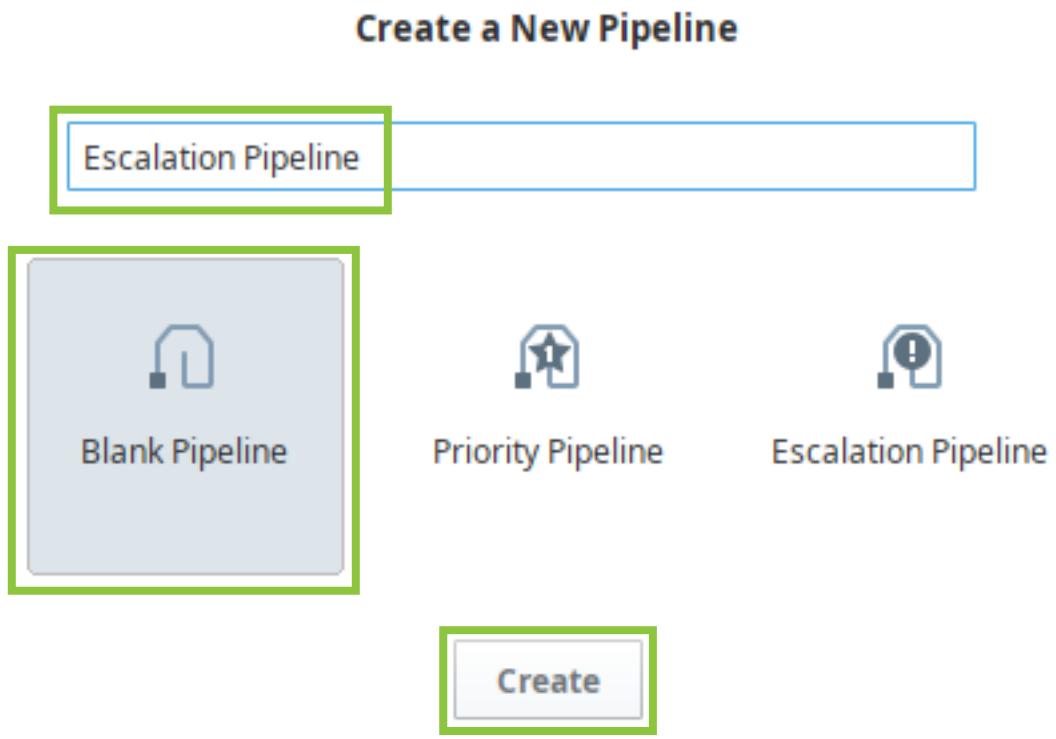
Now that we have created a simple pipeline to notify a group of users, let's create a more complex pipeline.

In this scenario, we are going to notify our operators roster when an alarm becomes active. We will then give them a chance to acknowledge the alarm. If they do not acknowledge the alarm, we will notify them a second, then a third time. If they still have not acknowledged the alarm after the third notification, we are going to escalate the issue, and notify the managers roster.

This flowchart shows the logic of the pipeline, to give you an idea of what we are building.



1. Click **Alarm Notification Pipelines** in the Project Browser.
2. Enter **Escalation Pipeline** in the Name of the pipeline field.
3. Click **Blank Pipeline**.
4. Click **Create**.



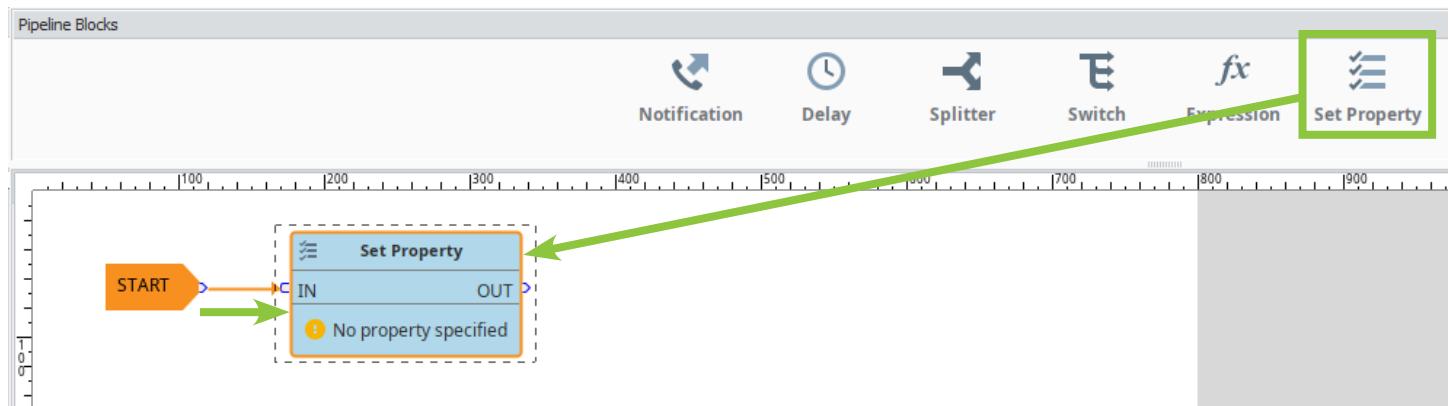
Create a Counter

First, we will need a counter to keep track of how many times we have notified our users.

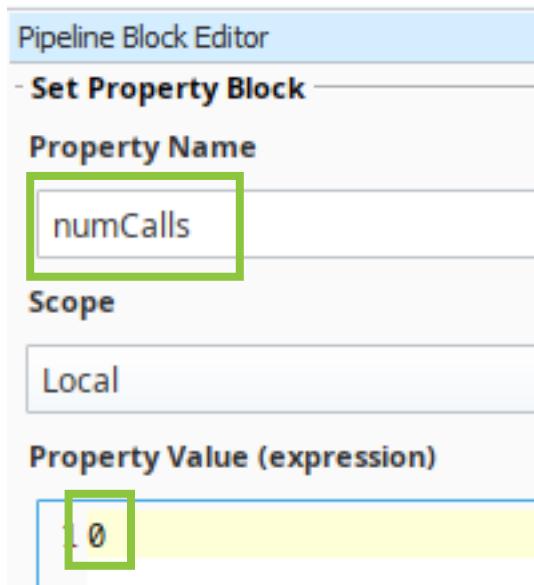
1. Drag a **Set Property** block to the pipeline window.

This block creates a variable that exists only within the pipeline.

2. Press and hold your mouse over the output pin on **START**.
3. Click and drag to the input pin on **Set Property**.



4. Enter **numCalls** in the Property Name field in the Pipeline Block Editor.
5. Enter **0** in Property Value (expression).

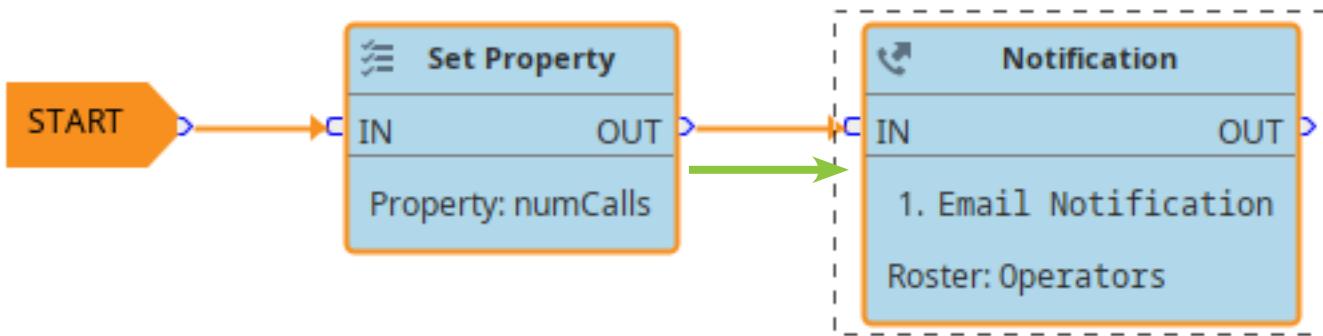


Note: Pay close attention to the spelling and capitalization of the Property Name. We will need to enter this manually later, and will need it to exactly match.

Add a Notification Block

Now that we have initialized the property to track how many times our operators have been notified, we need to send the first notification.

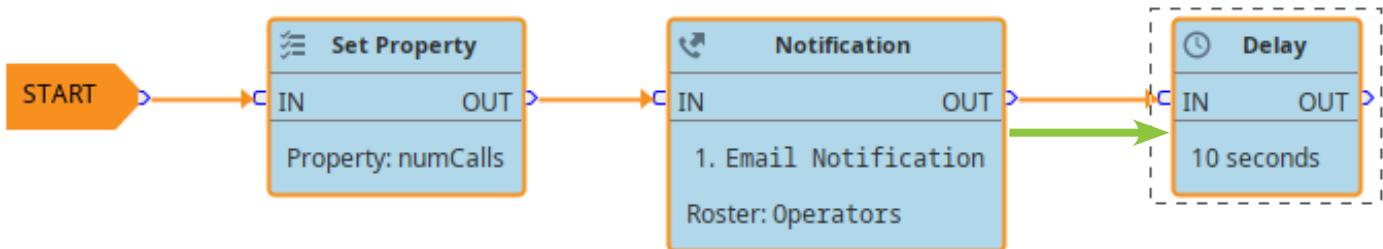
1. Drag a **Notification** block to the pipeline window.
2. Click the **Notification** block.
3. Select **Email Notification** from the Choose One list in the Pipeline Block editor.
4. Click **Contacts**.
5. Select **Operators** from the On-call Rosters list.
6. Drag the **output pin** on the Set Property block to the **input pin** on the Notification block.



Add a Delay

We want to give our operators some time to acknowledge the alarm before sending another notification.

1. Drag a **Delay** block into the pipeline.
2. Drag the **output pin** on the Notification block to the **input pin** on the Delay block.
3. Click the **Delay** block.
4. Enter **10** in the Delay (sec) field.



Note: Obviously, 10 seconds is not a reasonable time frame to allow operators to acknowledge the alarm. We are setting a very low value here so that we can quickly test and make see how this pipeline will work.

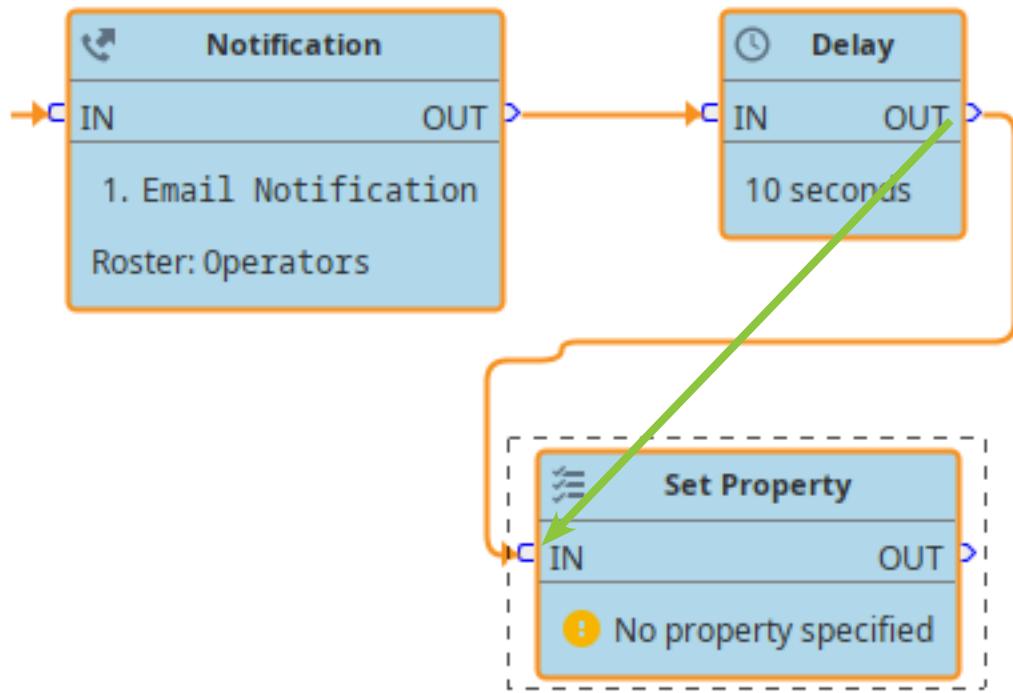
Increment the Counter Property

If operators do not acknowledge the alarm, we want to notify them again. But before we do that, we need to increment the value of our counter, and then test that value to see how many times we have sent notifications and escalate it needed.

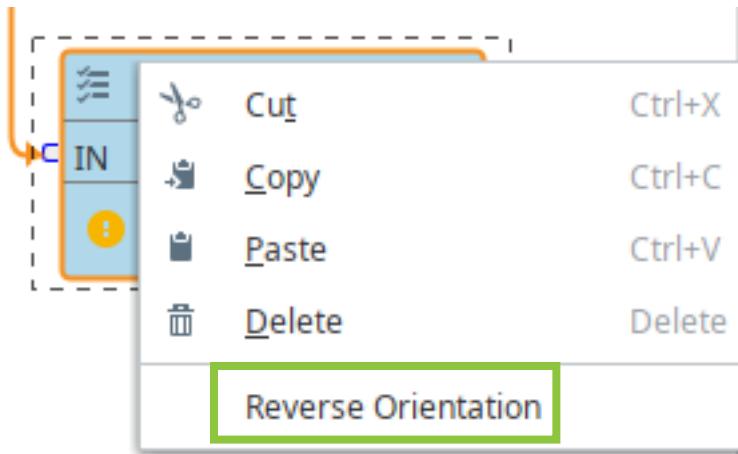
We are also going to add some steps to ensure that our pipeline remains easily readable. This is particularly important when we are dealing with complex pipelines. In this case, we will end up with a loop in our pipeline, so we want to set up our pipelines to visually represent that loop.

1. Drag a **Set Property** block to the pipeline window.
2. Position the **Set Property** block below the **Delay** block.
3. Drag the **output pin** of the Delay block to the **input pin** on the Set Property block.

You'll see that this causes the connecting line to wrap around the block. We don't want that.



4. Right-click the **Set Property** block.
5. Click **Reverse Orientation**.



6. Click the **Set Property** block.
7. Enter **numCalls** in the Property Name field.

Note: You must ensure that you type this property precisely like you did the first time, as we want to change the value of the existing property, not create a new one. Pay close attention the capitalization.

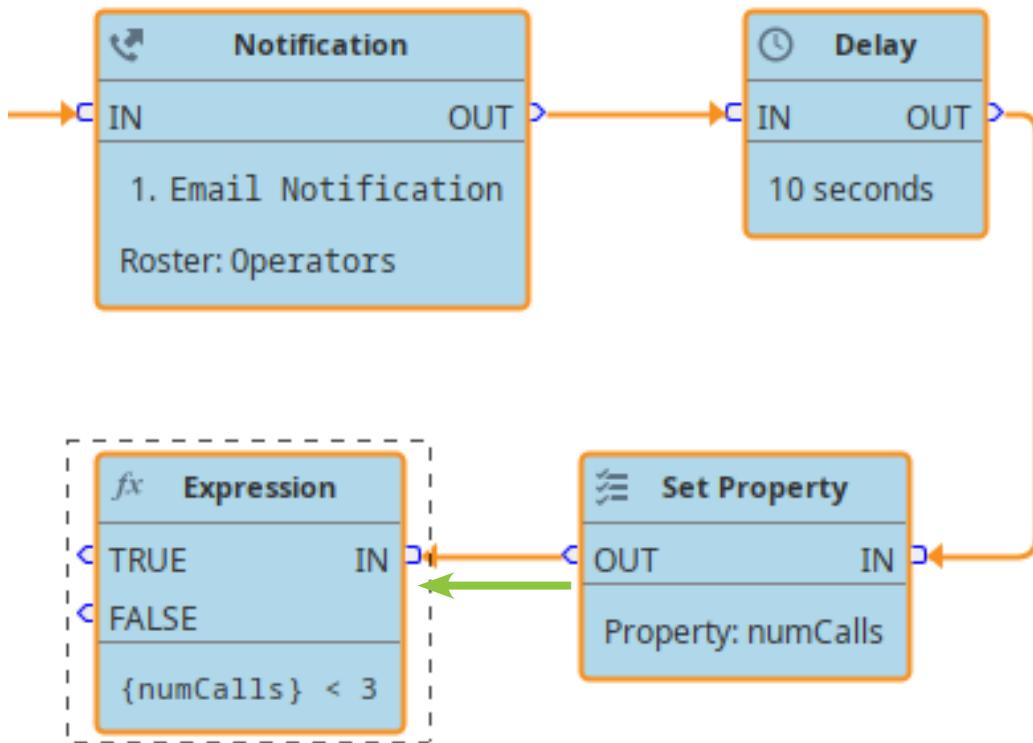
8. Enter **{numCalls} + 1** in the Property Value (expression) field.

This expression takes the current value of numCalls and adds 1 to it.

Test numCalls

Next, we need to test the value of our property. If it is less than 3, then we will notify the same roster again. If it is 3, we will escalate the pipeline.

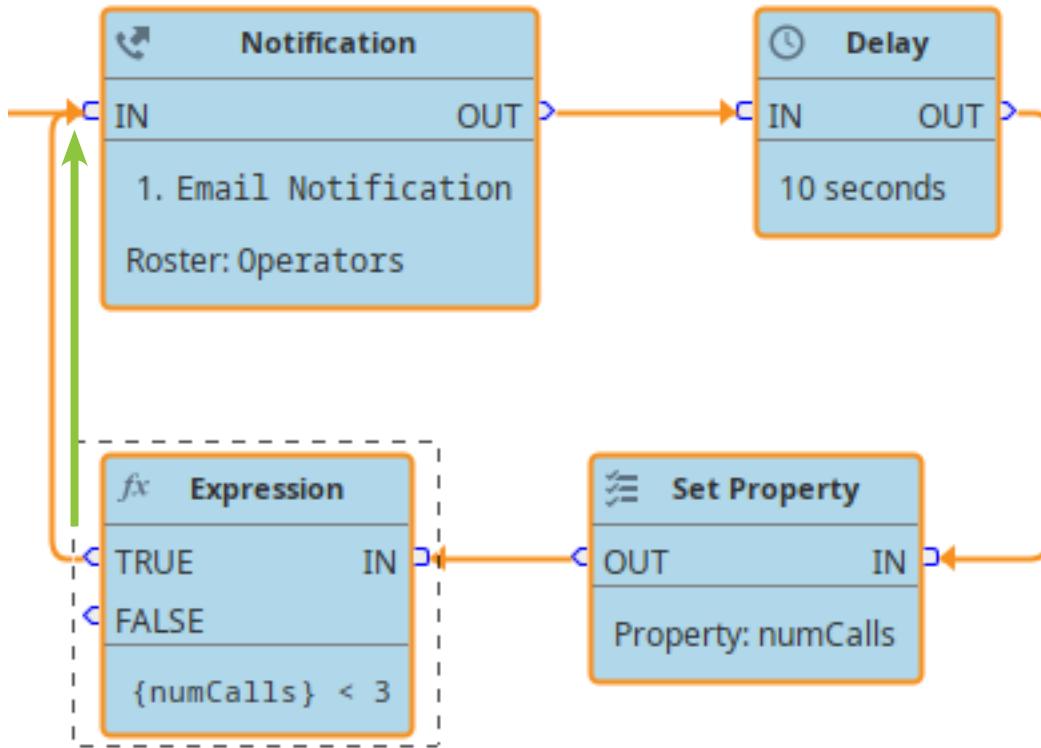
1. Drag in an **Expression** block into the pipeline.
2. Right-click the **Expression block**.
3. Click **Reverse Orientation**.
4. Drag the **output pin** of the Set Property block to the **input pin** on the Expression block.



5. Enter `{numCalls} < 3` in the **Expression Block**.

If the expression is false, meaning that we have sent fewer than 3 notifications, we want to restart our loop.

6. Drag the **output pin** next to TRUE on the Expression Block to the **input pin** on the original Notification block.



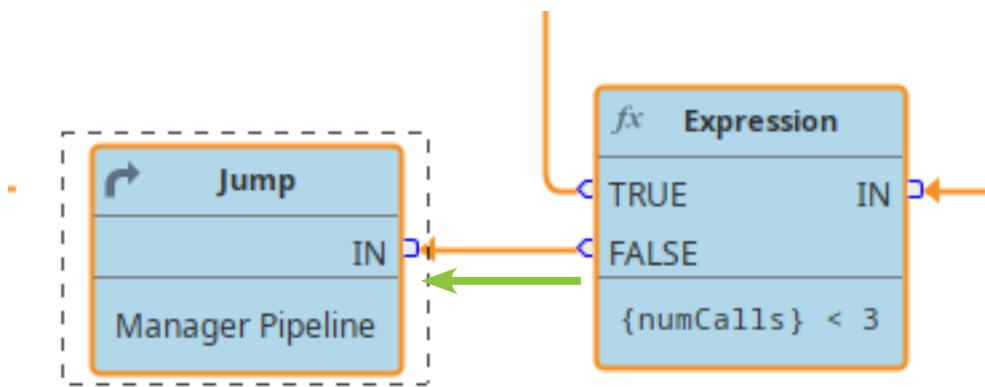
Note: You can connect blocks to as many other blocks as you need.

Jump to the Other Pipeline

If our expression is false, meaning that we have sent 3 notifications, we want to escalate this pipeline to notify our managers. But recall that we already have a notification pipeline that sends emails to that managers roster: our Manager pipeline. Rather than recreating that, we can simply connect to it.

1. Drag a **Jump** block to the pipeline window.
2. Right-click the **Jump** pipeline.
3. Click **Reverse Orientation**.
4. Drag the **output pin** next to FALSE on the Expression block to the **input pin** on the Jump block.
5. Click the **Jump** block.
6. Select **Manager Pipeline** from the Destination Pipeline list.
7. Click **File**.

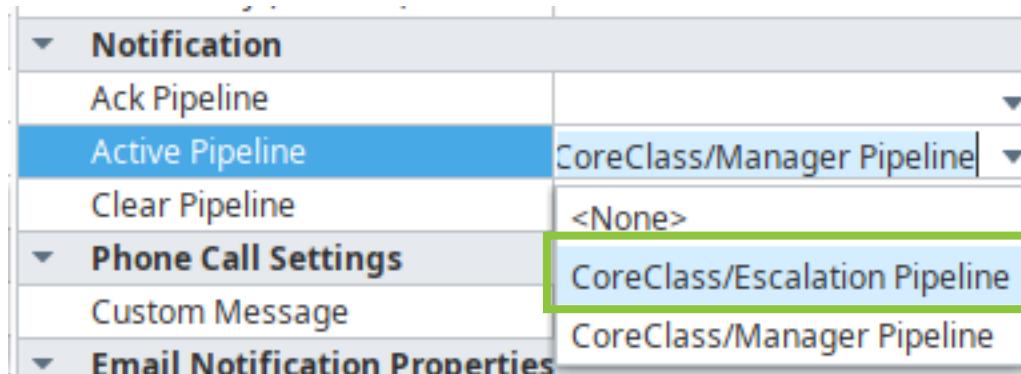
8. Click **Save All**.



Apply and Test the New Pipeline

Now that we have the pipeline created, lets replace the Manager pipeline on our Tag with this one so we can test it.

1. Expand **GenSim** on the Tag Browser.
2. Expand **Writable**.
3. Double-click **WritableBoolean1**.
4. Click **Alarms**.
5. Click **Test Alarm**.
6. Scroll to the **Notification** settings.
7. Select **CoreClass/Escalation Pipeline** from the Active Pipeline list.



8. Click **OK**.

Ch 16. Alarm Notification

9. Click **File**.
10. Click **Save All**.
11. Enable **WritableBoolean1** in the Tag Browser.
12. Switch back to **PaperCut**.

The screenshot shows the Papercut SMTP application interface. On the left, a sidebar lists several alarm notifications from the Ignition system. On the right, the main window displays an email message with the following details:

From	default@example.com
To	manager@acme.com
Date	8/30/2024 6:27:12 PM +00:00
Subject	Ignition Alarm Notification

The message body contains the text: "At 06:27:12, alarm "Test Alarm" at "" transitioned to Active." and a link: http://localhost:8088/web/ack/z_d6ld-xAwKq3ZK09-y.jwq4UY_TQBCeQDVTiMSxhCn7-. At the bottom, there are buttons for FORWARD, DELETE (1), and DELETE ALL, along with the application logo and URL.

You should get two new emails right away, since both our manager and our operator users are in the Operators roster. 10 seconds later, you will get two more. 10 seconds after that, you'll get two more. However, 10 seconds later, you'll get three emails: the pipeline has now escalated the issue and send messages to admin, supervisor, and manager.

Dropout Condition

You might be asking yourself, “how does the pipeline know if an alarm has been acknowledged?” Let’s first see this in action, then discuss how it works.

1. Click **Delete All** in PaperCut.
2. Switch back to **Designer**.
3. Enable WriteableBoolean1 in the Tag Browser.
4. Switch back to **PaperCut**.

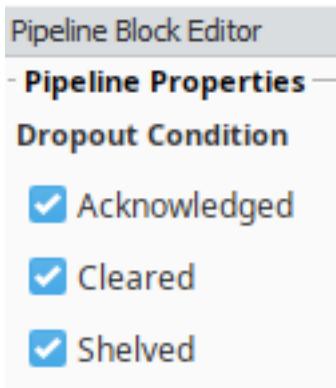
5. Click one of the emails.
6. Click the **link**.
7. Enter a **note**.
8. Click **Acknowledge**.
9. Return to **PaperCut**.

You will see that no further emails are received. Acknowledging the alarm causes the alarm to drop out of the pipeline.

Configure Dropout Conditions

There are three possible dropout conditions for each pipeline: Acknowledged, Cleared, and Shelved. These dictate when alarms fall out of the pipeline. If any of the selected conditions become true for an event in a pipeline, it will “drop out” entirely, no matter where it is at. No further action will be taken for that event.

1. Return to **Designer**.
2. Click in a blank area of the pipeline window so that no blocks are selected.



With nothing selected, the Pipeline Block Editor shows the dropout conditions. These are evaluated in order from top to bottom. Therefore, once an alarm is Acknowledged, it drops out of the pipeline and no further notifications will be sent.

Popup Windows

In previous chapters, we created a template to show Tag data for our motors. We also made the template dynamic, so that it could look at data for any single motor, as opposed to being tied to a single one. In this chapter, we'll explore a similar task, using a Popup window to dynamically show more detailed information for a given motor.

Creating a Popup Window

In this exercise, we'll be configuring a parameterized Popup window. That is, a Popup window that takes a parameter, and uses that parameter to decide which of our items we want to show data for. This approach, notably, will eliminate the need to create a new popup window for each motor.

Create a Detailed Motor Popup Window

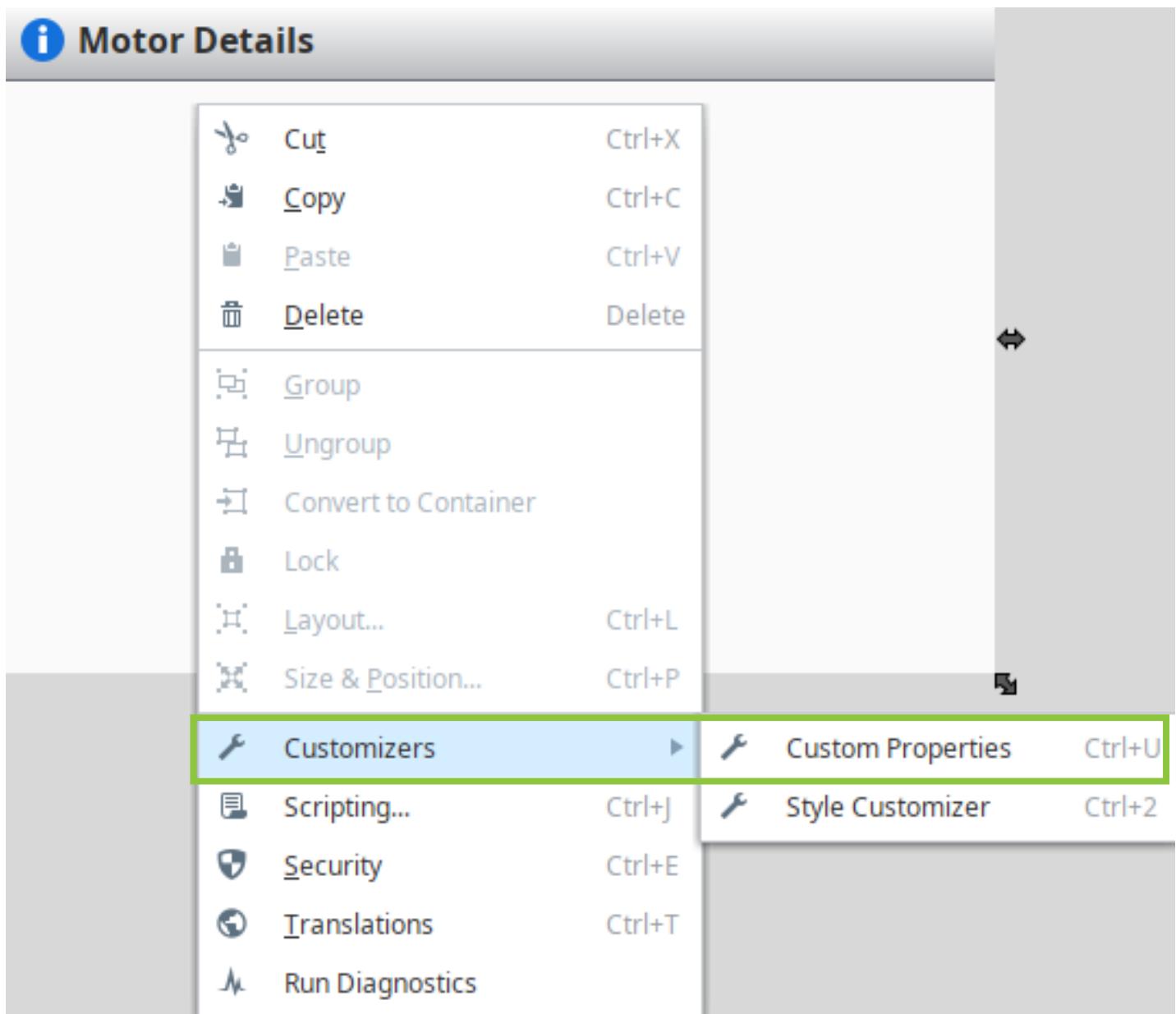
The project template we are using includes a blank popup window. We are going to duplicate that for ours. Even though it is in a folder called **Popups**, it is the Window's settings that make it a popup. See "Window Types" on page 74.

1. Open **Designer**.
2. Click **Vision**.
3. Expand **Windows**.
4. Expand **Popups**.
5. Right-click **Popup**.
6. Click **Duplicate**.
7. Right-click **Popup (1)**.
8. Click **Rename**.
9. Enter **Motor Details**.
10. Double-click the header **Label** component.
11. Enter **Motor Details**.

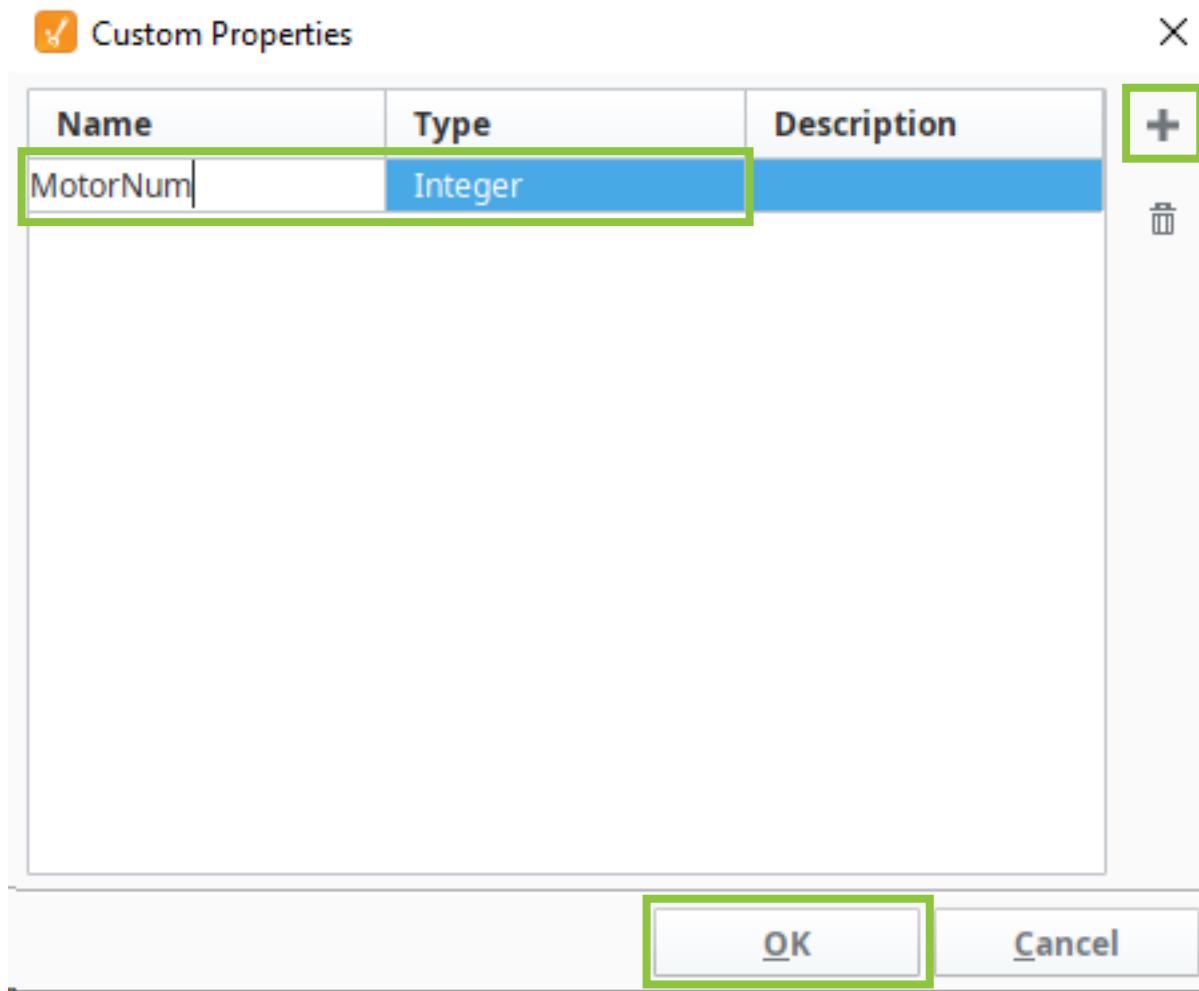
Add a Parameter to the Popup

In order for the popup to dynamically display information, it will need one or more parameters—data to be passed into the popup from whatever triggered it that the components on the popup can use to change the information they display.

1. Right-click the **background** of the popup.
2. Select **Customizers**
3. Click **Custom Properties**.



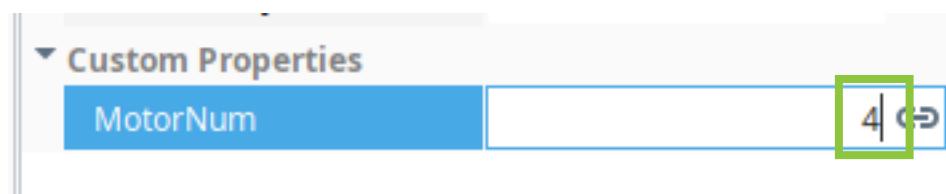
4. Click .
5. Enter **MotorNum** in the Name field.
6. Select **Integer** as the Type.
7. Click **OK**.



If you examine the Vision Properties for the Root Container, you will now see our new parameter in the **Custom Properties** section. Pay attention, however, to its default value: because the dialog box used to create the property does not give us a way to set a default value, it will be set to 0. While this is technically a valid value, we do not have a Motor 0, so unless we change it, our bindings will appear to be broken even if they are correct.

8. Click **0**.

9. Enter **4**.

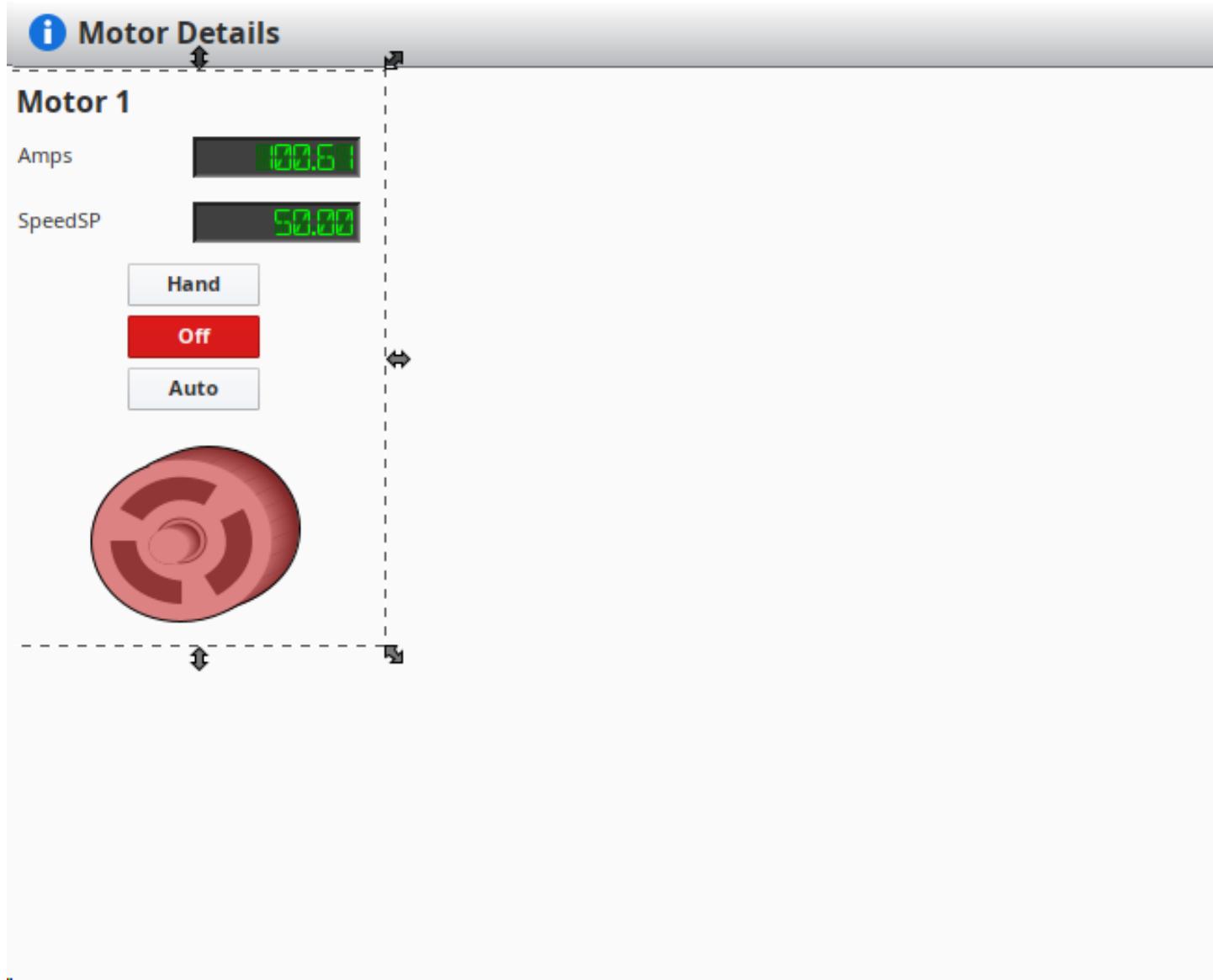


This sets our property to a number that matches one of our motors, so bindings will now work correctly as we create them.

Add Components to the Popup

You can add components to the Popup in the same way that you add them to a Main Window.

1. Drag the **Motor Template** to the Popup Window.

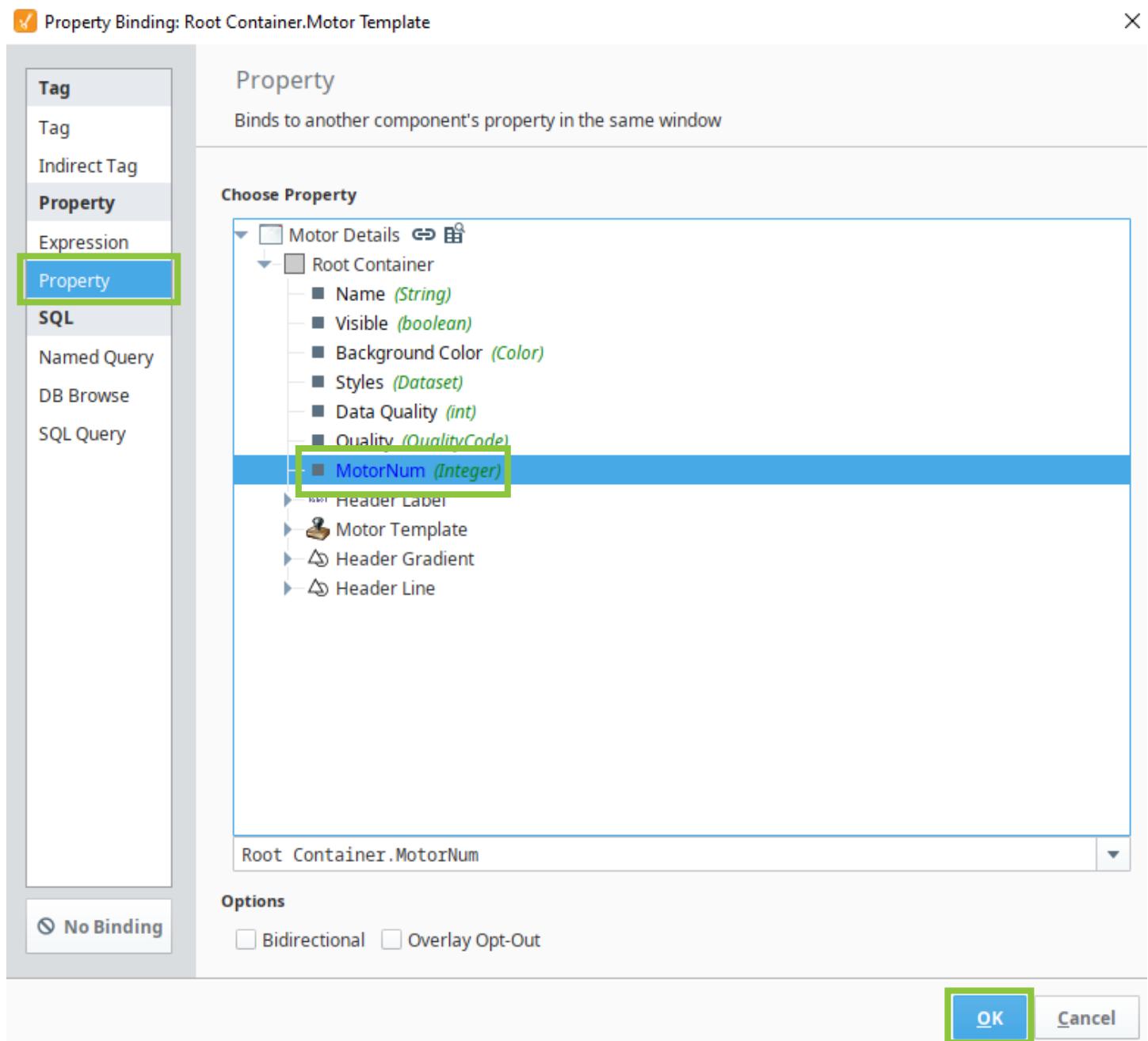


Note: Be sure the Motor Template is not open in Designer.

The Motor Template takes a parameter—the motor number—to display the details of a particular motor. Our Popup also has a parameter designed to represent the same thing.

So, binding the template's parameter to the popup's will allow us to have the template display details for whichever motor the popup is showing.

2. Click the **template**.
3. Click  on the **MotorNumber** property.
4. Click **Property**.
5. Expand **Root Container**, if necessary.
6. Click **MotorNum**.
7. Click **OK**.



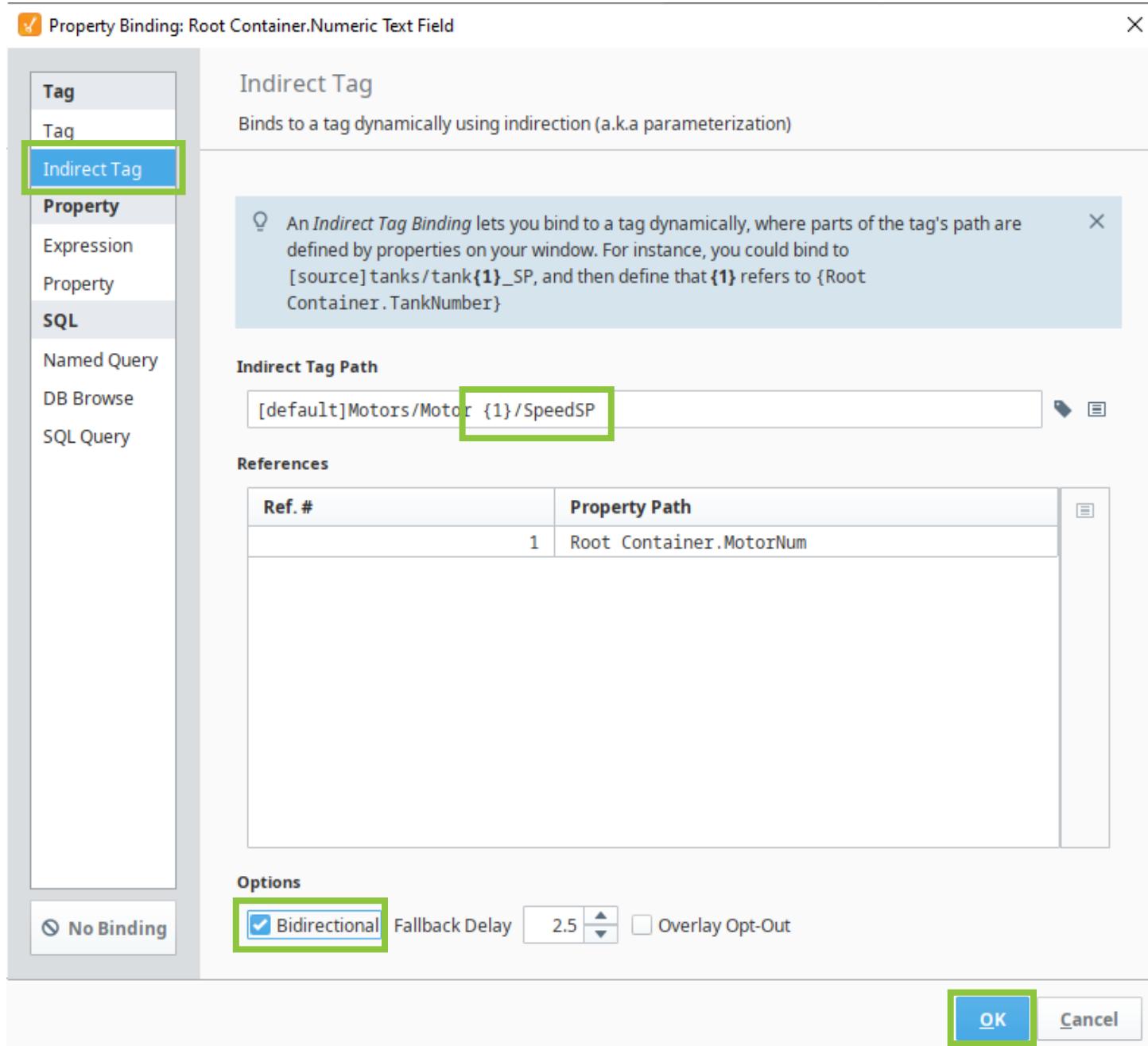
The template will update to display information for Motor 4. You can change the popup's MotorNum property to something else to ensure the binding is correct.

Now, let's add a few more components to the Popup.

1. Expand the **Components Palette**.
2. Click in the **Filter** box.
3. Enter **numeric**.
4. Drag a **Numeric Text Field** to the window.

We want to bind this text field to the SpeedSP Tag in our Motors. We will need an indirect Tag binding to do this.

5. Click  on the **Value (Integer)** property.
6. Click **Indirect Tag**.
7. Click .
8. Expand **Tags**.
9. Expand **Motors**.
10. Expand **Motor 1**.
11. Click **SpeedSP**.
12. Click **OK**.
13. Select **1** in the Tag path.
14. Press **Delete**.
15. Click .
16. Expand **Root Container**.
17. Click **MotorNum**.
18. Click **OK**.
19. Check **Bidirectional**.
20. Click **OK**.

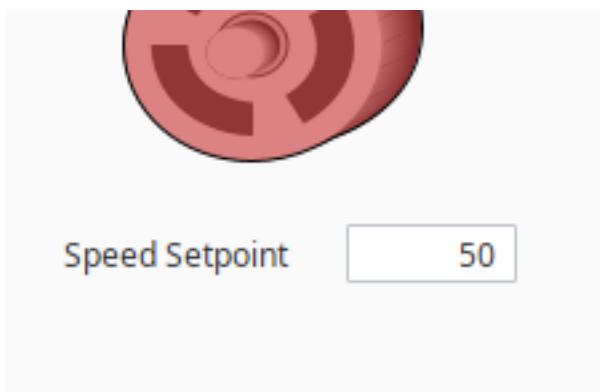


The numeric text field will update to show the value of the Motor 4 SpeedSP (assuming your parameter is still set to 4.)

We do not want unlabeled fields on our interfaces, so let's quickly fix that.

1. Expand the **Components Palette**.
2. Click in the Filter box.
3. Click  to clear to previous filter.
4. Enter **label**.

5. Drag a **Label** to the window.
6. Double-click the Label.
7. Type **Speed Setpoint**.



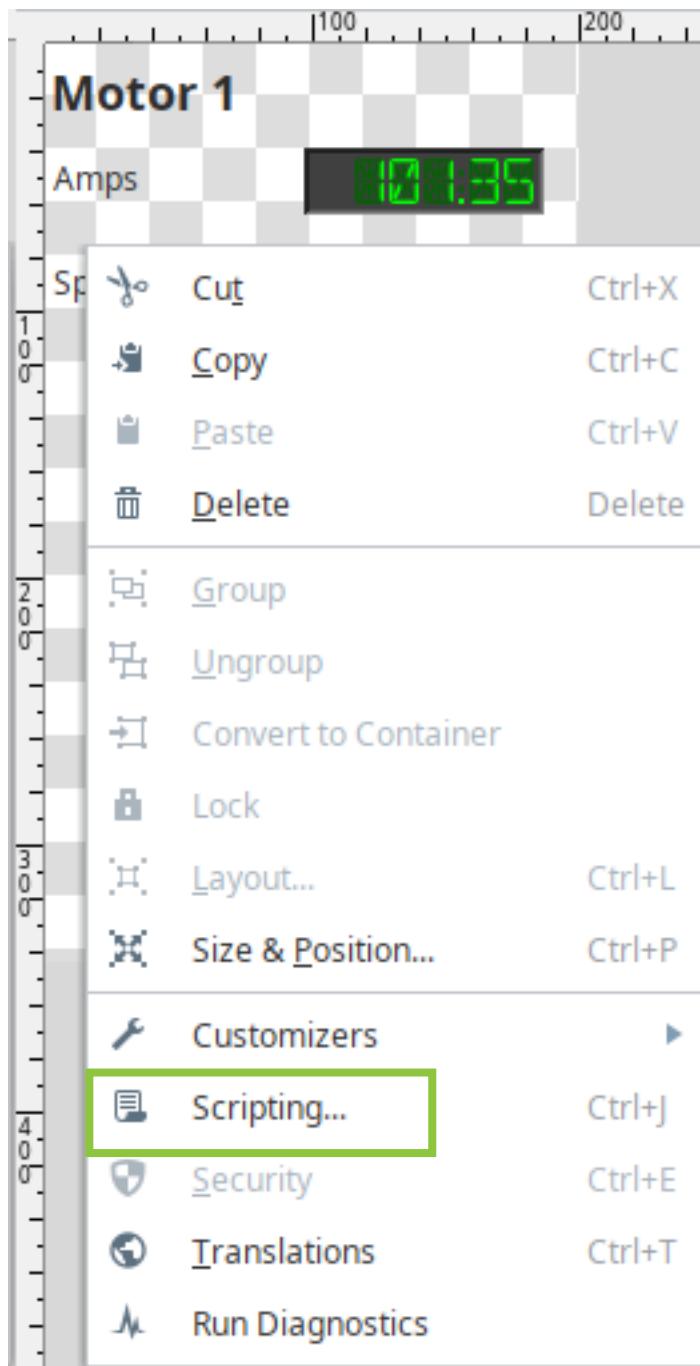
Opening a Popup

Earlier in class, we created a Main Window that displays multiple copies of our Motor Template. We want users to be able to click on one of those instances of the template and open the Popup. To do this, we need to add a navigation script to the background of the popup.

1. Double-click **Motor Template**.
2. Right-click in the **background** of the template.

Note: You may need to try several locations to find a spot on the template that is not covered by one of the components. If you cannot find a spot, you can just right-click on the template in the Project Browser.

3. Click **Scripting**.

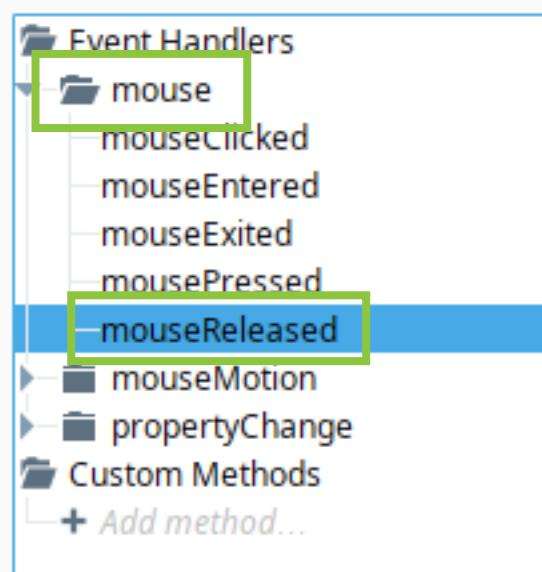


The Component Scripting dialog box has a lot of features, and we will spend considerably more time on it later. For now, we only need the first tab.

All scripts are based on events. Again, we will discuss this in more detail in the chapter on scripting, but for now, we can simply select the event we need to open a popup. Then, we can complete the rest of the dialog box.

1. Expand **mouse** under **Event Handlers**.
2. Click **mouseReleased**.

Component Scripting [Motor Temp]



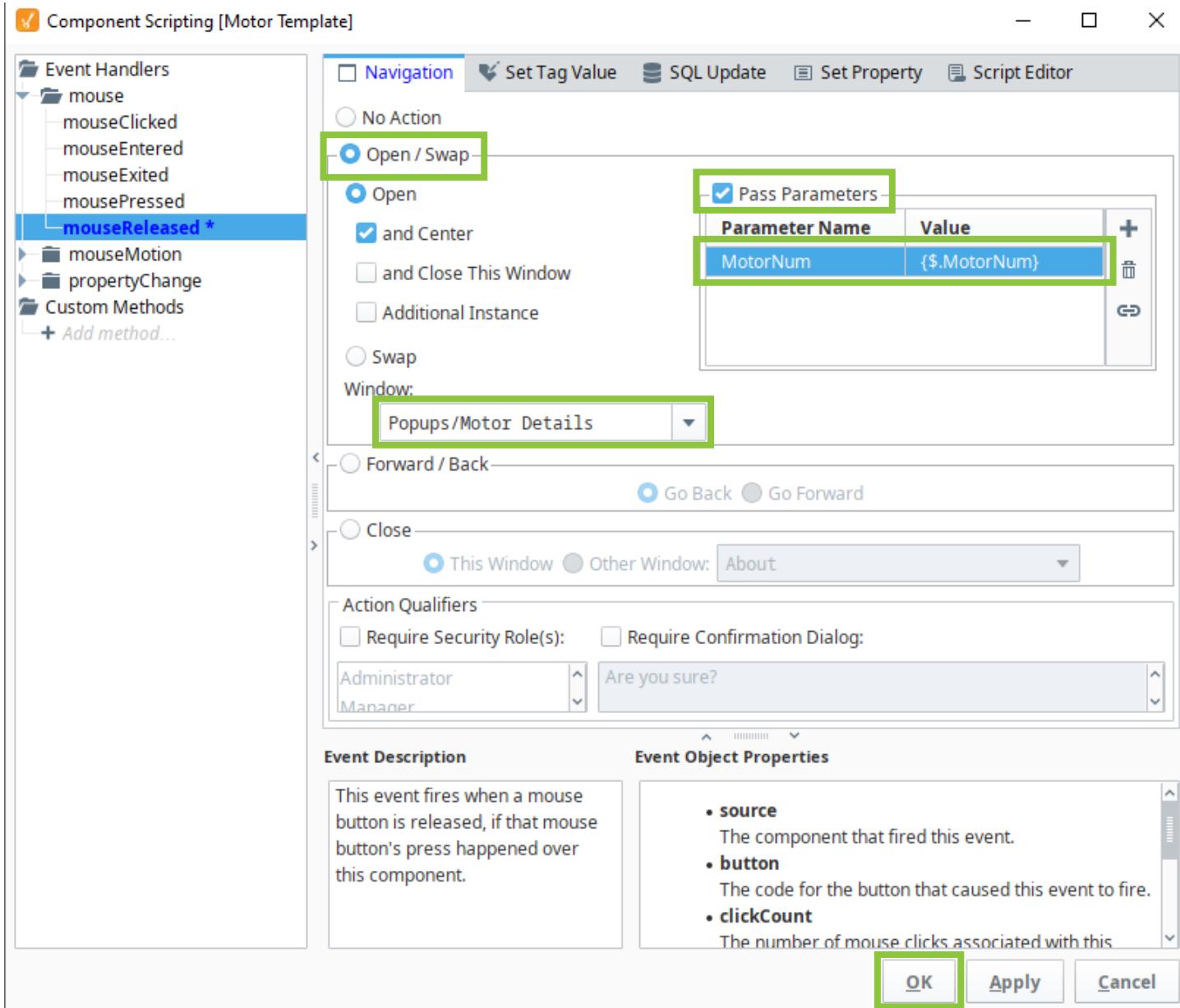
3. Click **Open/Swap**.
4. Select **Popups/Motor Popup** from the **Window** list.

When we click on an instance of the template in the Window, we need to pass the parameter from the template into the popup, which will then become the popup's parameter value.

5. Click **Pass Parameters**.
6. Click .
7. Select **MotorNum** from the **Parameter Name** list.

Note: This list is populated based on the window that you chose in step 7.

8. Double-click the **Value** field.
9. Click  for use as parameter value.
10. Click **MotorNumber**.
11. Click **OK**.
12. Click **OK**.

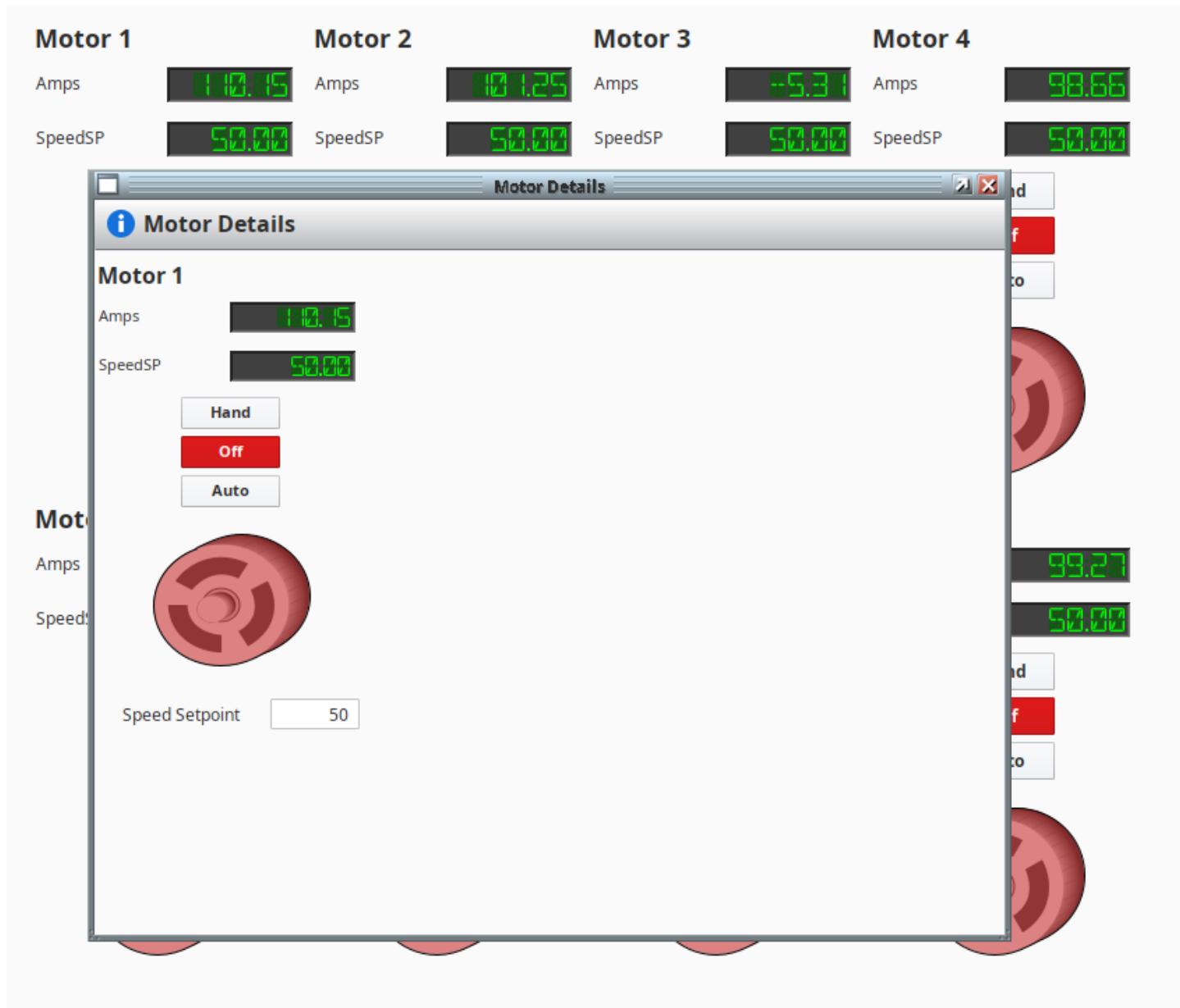


Even though we have not finished working on our popup, we can go ahead and test and make sure the scripting functionality, and in particular the passing of the parameter, is working correctly.

13. Click **File**.
14. Click **Save All**.
15. Click **Tools**.
16. Click **Launch Project**.
17. Click **Launch (windowed)**.
18. **Log into** the client. It does not matter what user account you use.

19. Click **Motor View**.

20. Click on any **motor**.



The popup will appear, and the instance of the template inside the popup will be set to the same template instance you clicked on.

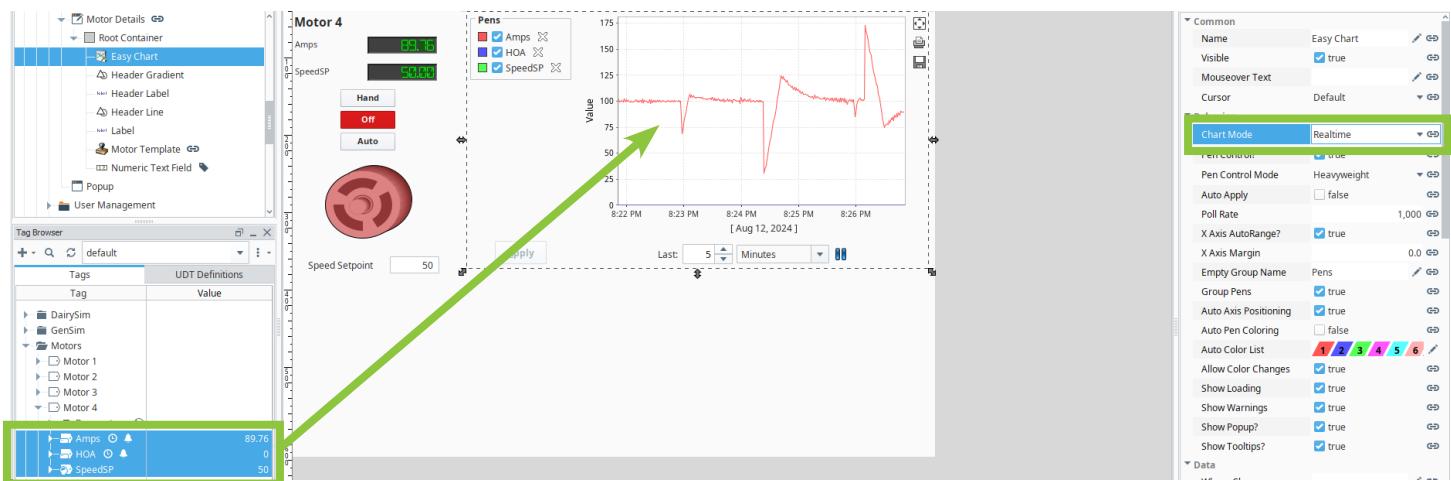
21. Click on **any other motor** on the Main Window.

The popup will update to show the correct motor.

Creating an Indirect Easy Chart

We are storing Tag history on our motors, and want to display that data on the popup. An ideal component for displaying historical data is the Easy Chart. (See “Use an Easy Chart” on page 196 for more information.)

1. Return to the **Motor Popup**.
2. Click in the **background** of the popup so that no components are selected.
3. Resize the **popup**.
4. Expand the **Components Palette**.
5. Click in the **Filter** box.
6. Click  to clear to previous filter.
7. Enter **easy**.
8. Drag an **Easy Chart** to the popup.
9. Size and position the **Easy Chart** so that it is roughly the same height as the template and covers the area to the right of the template.
10. Expand **Motors** in the **Tag Browser**.
11. Expand **Motor 4**.
12. Click **Amps**.
13. Shift-click **HOA**.
14. Drag all three Tags to the **Easy Chart**.
15. Select **Realtime** from the list on the **Chart Mode** property.



Customize the Chart

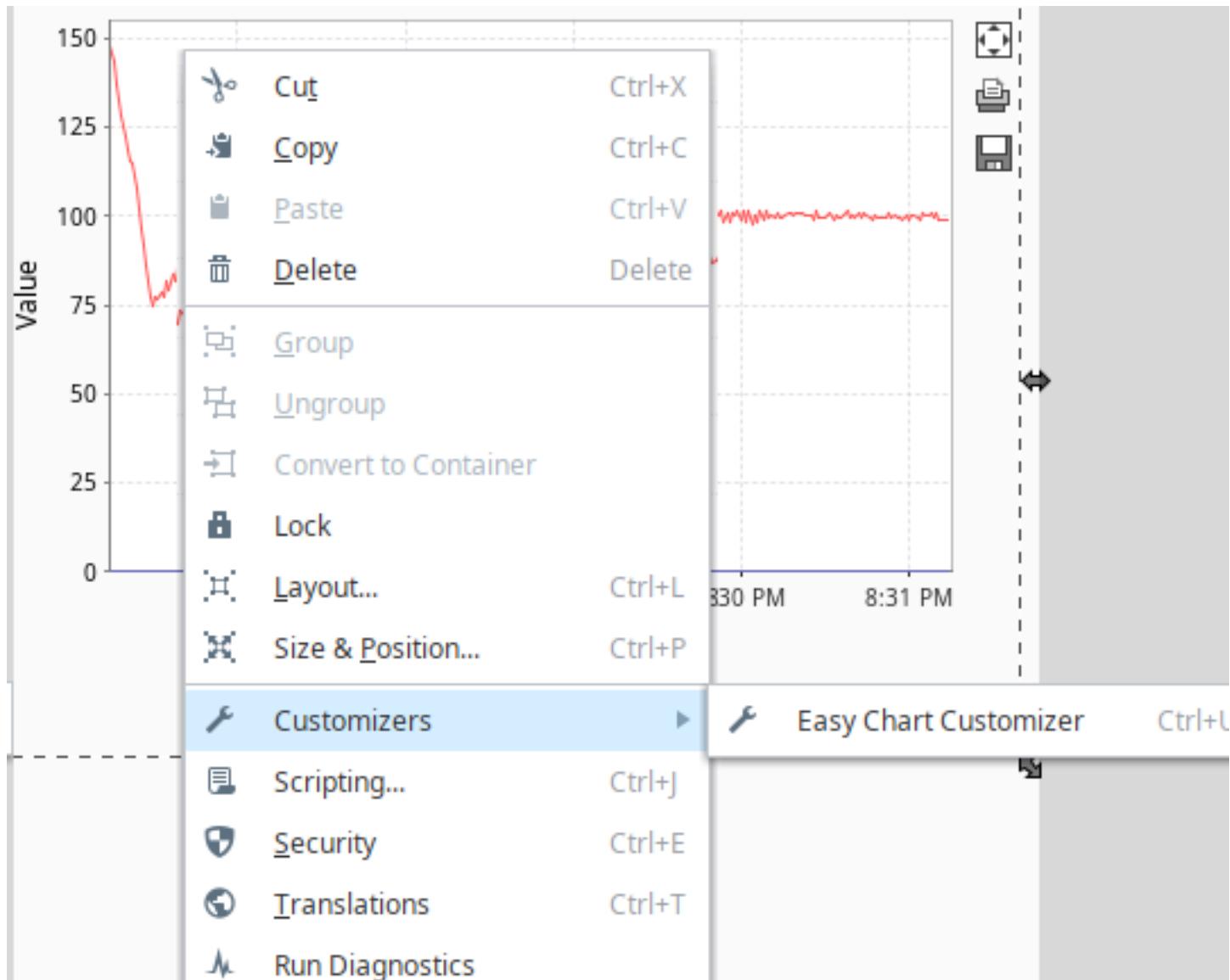
The three points of data, or pens, that we are displaying on the chart have radically different data ranges. On the one hand, the HOA Tag is strictly limited to 0, 1 and 2. The Amps, however, are generally hovering around 100, but occasionally spike much higher or lower. And while the SpeedSP Tag is relatively static, its value is going to be closer to the Amps than the HOA.

In addition, the Amps and SpeedSP values are directly related: the SpeedSP is used to trigger alarms when the Amps get too high. But the HOA isn't really related to those other two, other than being a Tag reported by the same device.

For these reasons, it doesn't make a lot of sense to display all three values on the same plot in the chart. It would make a lot more sense—and result in a much more readable chart—if the Amps and SpeedSP were on one plot, and the HOA on a separate plot.

Thankfully, the Easy Chart provides a range of customization possibilities.

1. Right-click the **Easy Chart**.
2. Select **Customizers**.
3. Click **Easy Chart Customizer**.



4. Click **Subplots**.
5. Click **+**.
6. Double-click **Relative Weight** on **Plot 1**.
7. Enter **3**.

The screenshot shows the 'Subplots' tab settings. There are two plots listed:

Plot #	Relative Weight	Custom Background?	Background
Plot 1	3	<input type="checkbox"/>	<input type="button"/> <input type="button"/> <input type="button"/>
Plot 2		<input type="checkbox"/>	<input type="button"/> <input type="button"/> <input type="button"/>

A green box highlights the 'Relative Weight' input field for Plot 1, which contains the value '3'. Another green box highlights the '+' button at the bottom right of the table, used to add more plots.

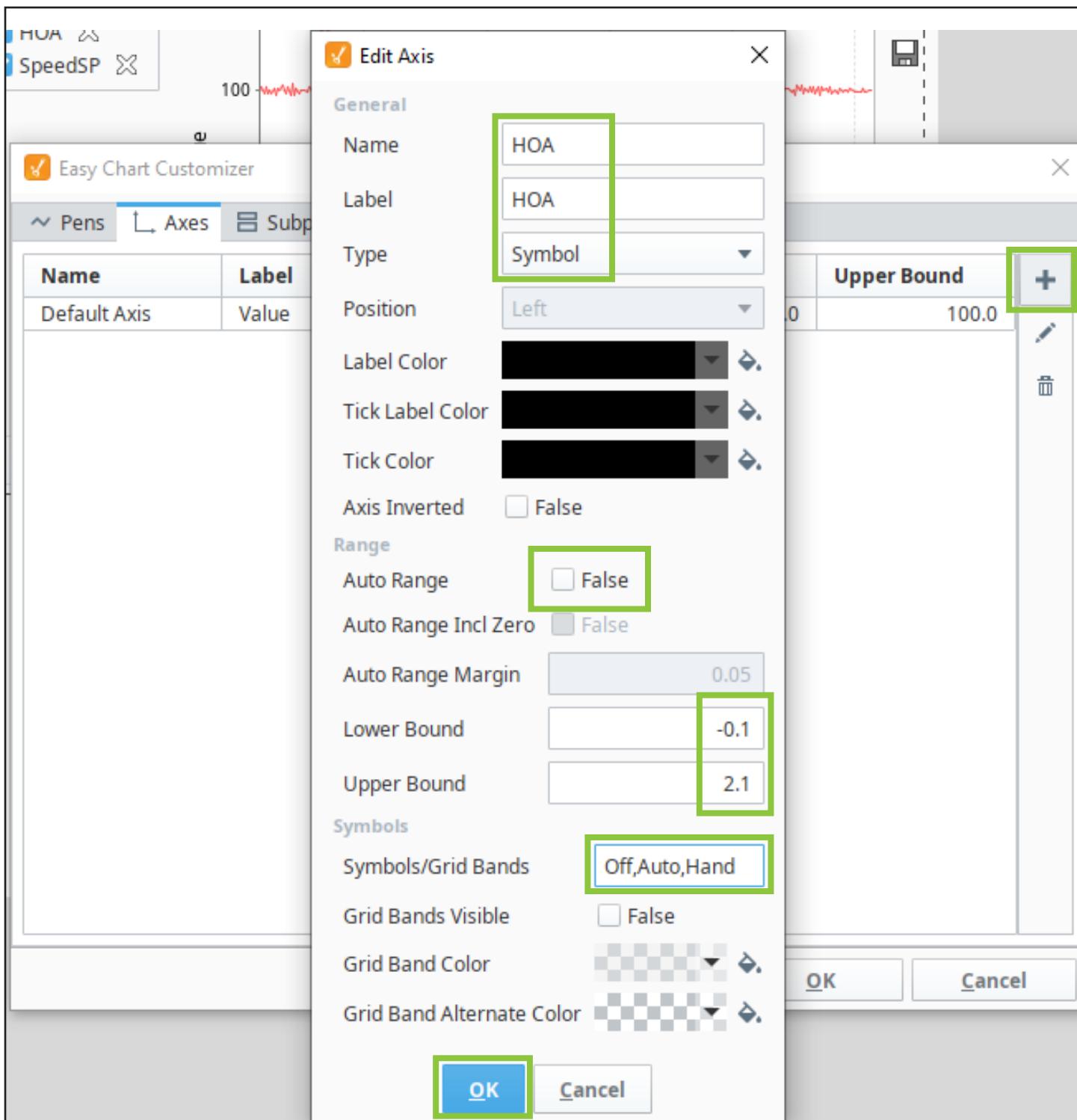
The Relative Weight setting allots the vertical space of the Easy Chart proportionally. By setting Plot 1 to 3 and leaving Plot 2 at 1, we are saying that we want Plot 1 to have 75% of the height of the chart, and Plot 2 to have the other 25%.

Next, we need to set up a custom vertical axis for the HOA Tag. As we know, its values are either 0, 1, or 2. We want to set the upper and lower bounds of the plot to slightly more than 2 and slightly less than 0 so that those values will be more visible. We also want to add custom labels so that it shows Off, Auto, and Hand instead of 0, 1, and 2.

1. Click **Axes**.
2. Click .
3. Enter **HOA** in the Name field.
4. Enter **HOA** in the Label field.
5. Select **Symbol** from the Type list.
6. Deselect **Auto Range**.
7. Enter **-0.1** in the Lower Bound field.
8. Enter **2.1** in the Upper Bound field.
9. Enter **Off,Auto,Hand** in the Symbols/Grid Bands field.

Note: Do not put spaces before or after the commas.

10. Click **OK**.



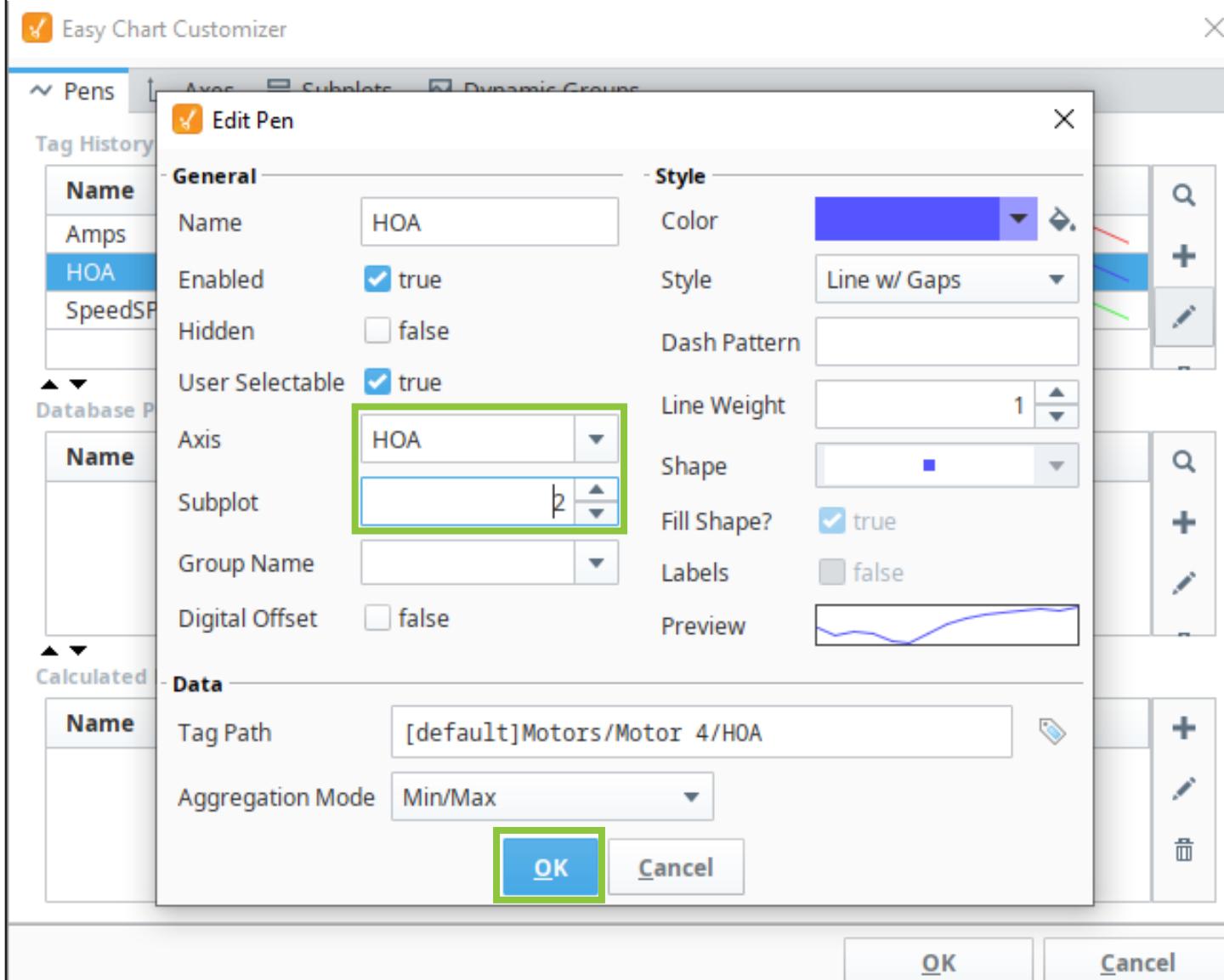
Finally, we have to configure the HOA Tag pen to use the subplot and axis we've configured.

1. Click **Pens**.
2. Click **HOA**.
3. Click .
4. Select **HOA** from the **Axis** list.

5. Set **Subplot** to **2**.

6. Click **OK**.

7. Click **OK**.



The chart will update so that now the HOA is on its own custom plot and the other two Tags are on their own plot.

Make the Chart Indirect

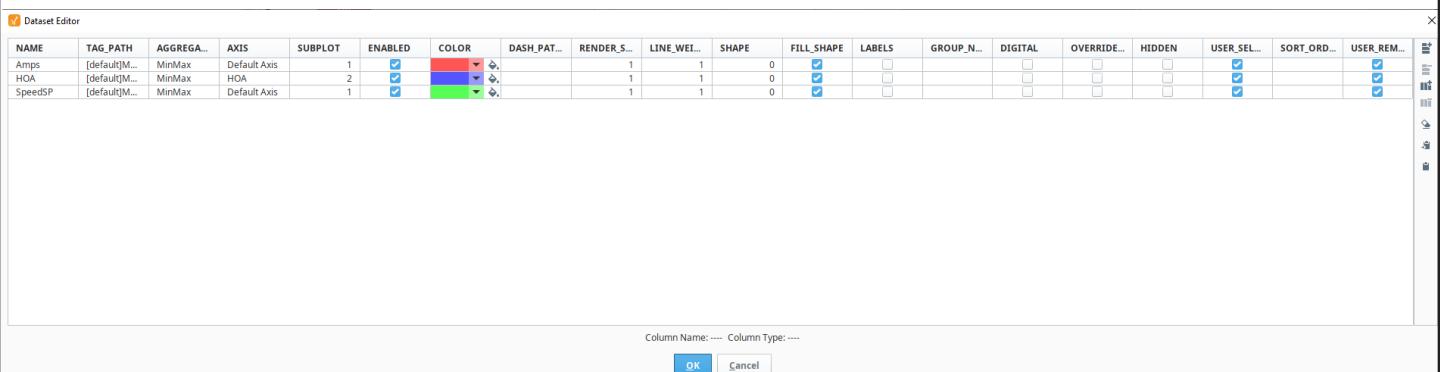
There's just one more problem, though. The chart can only display the values from Motor 4. We need to make its bindings indirect. But this isn't quite as straight-forward as it seems. Unlike other components we have explored, the Easy Chart doesn't have a

simple value property that the pens are bound to. So let's take a look at how the chart is getting its data in the first place.

1. Make sure the **chart** is selected.
2. Scroll to the **bottom** of the properties.
3. Click  on the Tag pens property.

The Easy Chart uses this dataset to populate itself. It has one row for each Tag you added to the chart, but it also has 20 columns, so this dialog box can be a bit challenging to work with at first.

4. Drag the **left edge** of the dialog box to the left to expand its width.
5. Repeat with the **right edge**.



NAME	TAG_PATH	AGGREGA...	AXIS	SUBPLOT	ENABLED	COLOR	DASH_PAT...	RENDER_S...	LINE_WEI...	SHAPE	FILL_SHAPE	LABELS	GROUP_N...	DIGITAL	OVERRIDE...	HIDDEN	USER_SEL...	SORT_ORD...	USER_Rem...
Amps	[default]M...	MinMax	Default Axis	1	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	1	1	<input checked="" type="checkbox"/>					<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	
HOA	[default]M...	MinMax	HOA	2	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	1	1	<input checked="" type="checkbox"/>					<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	
SpeedSP	[default]M...	MinMax	Default Axis	1	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	1	1	<input checked="" type="checkbox"/>					<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	

Column Name: ---- Column Type: ----

OK **Cancel**

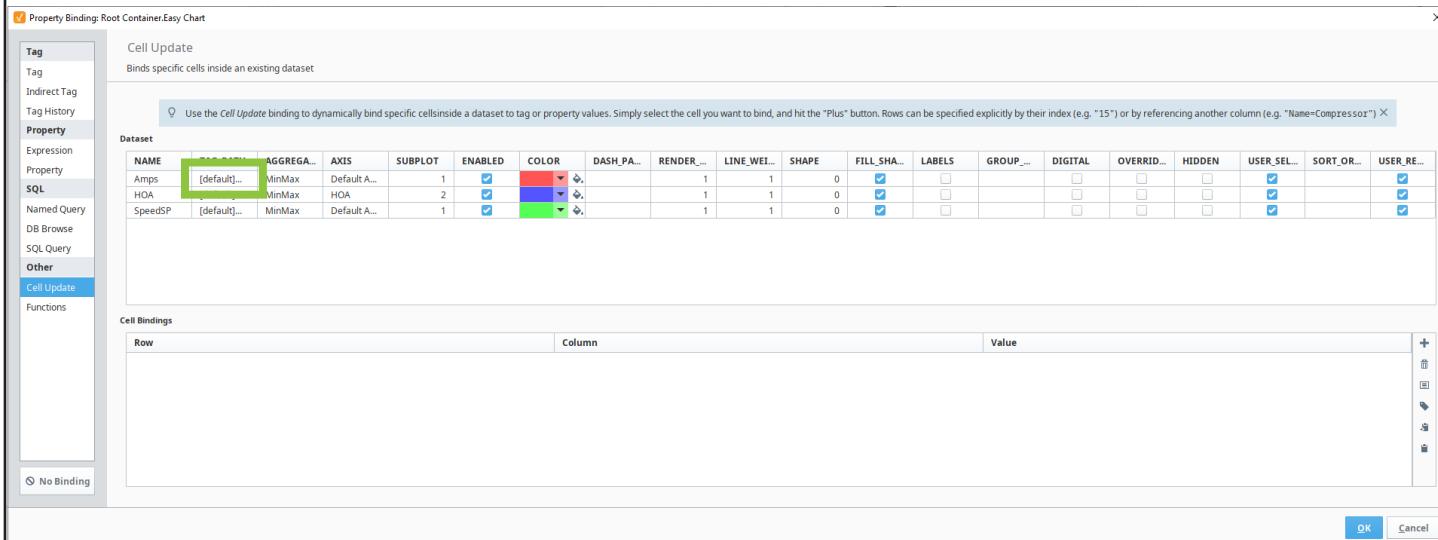
If your monitor is wide enough, you should not be able to see all 20 columns. Most of time are not important for our purposes, but one is: **TAG_PATH**. As you can see, this property shows the path to the Tag for each pen, and has the motor number hard-coded into the cell. So, in order to make this chart indirect, we need to use a special kind of binding, called a **Cell Update Binding**, to be able to go into this dataset and edit just the individual cell we want. But we can't do that from here; as always, we need to work with a binding.

1. Click **Cancel**.
2. Click  on the **Tag Pens** property.
3. Click **Cell Update**.

This shows the exact same table we saw a moment ago, but now we can add bindings to individual cells.

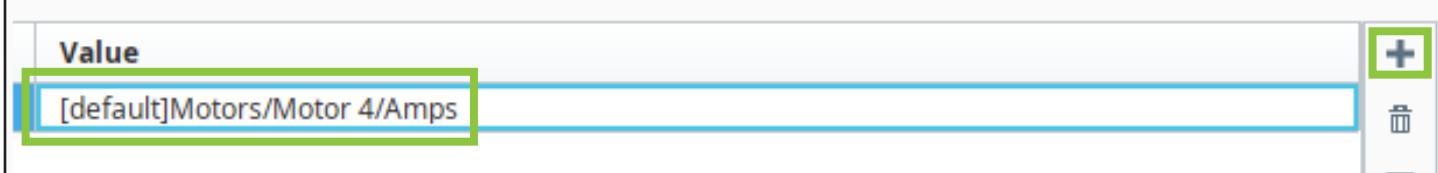
4. Drag the **left edge** of the dialog box to the left to expand its width.
5. Repeat with the **right edge**.

6. Click in the first **TAG_PATH** cell.



Normally, we would create a binding, then use the button to find the Tag we want, but that does not work for Cell Bindings. Instead, we need to copy and paste the Tag path from the table, then replace the hard-coded number with the parameter.

1. Press **Ctrl+C** on your keyboard to copy the selected path.
2. Click to add a binding.
3. Double-click in the **Value** cell.
4. Press **Ctrl+V** on your keyboard to paste the path.



5. Select **4** in the path.
6. Press **Delete**.
7. Click .
8. Expand **Root Container**.
9. Click **MotorNum**.
10. Click **OK**.

Now that our first Tag's path is dynamic, we can repeat those steps for the other two.

11. Click in the second **TAG_PATH** cell.
12. Press **Ctrl+C** on your keyboard to copy the selected path.
13. Click to add a binding.
14. Double-click in the **Value** cell.
15. Press **ctrl-v** on your keyboard to paste the path.
16. Select **4** in the path.
17. Press **Delete**.
18. Click .
19. Expand **Root Container**.
20. Click **MotorNum**.
21. Click **OK**.
22. Click in the third **TAG_PATH** cell.
23. Press **Ctrl+C** on your keyboard to copy the selected path.
24. Click to add a binding.
25. Double-click in the **Value** cell.
26. Press **Ctrl+V** on your keyboard to paste the path.
27. Select **4** in the path.
28. Press **Delete**.
29. Click .
30. Expand **Root Container**.
31. Click **MotorNum**.
32. Click **OK**.
33. Click **OK**.

Creating a Dynamic Alarm Display

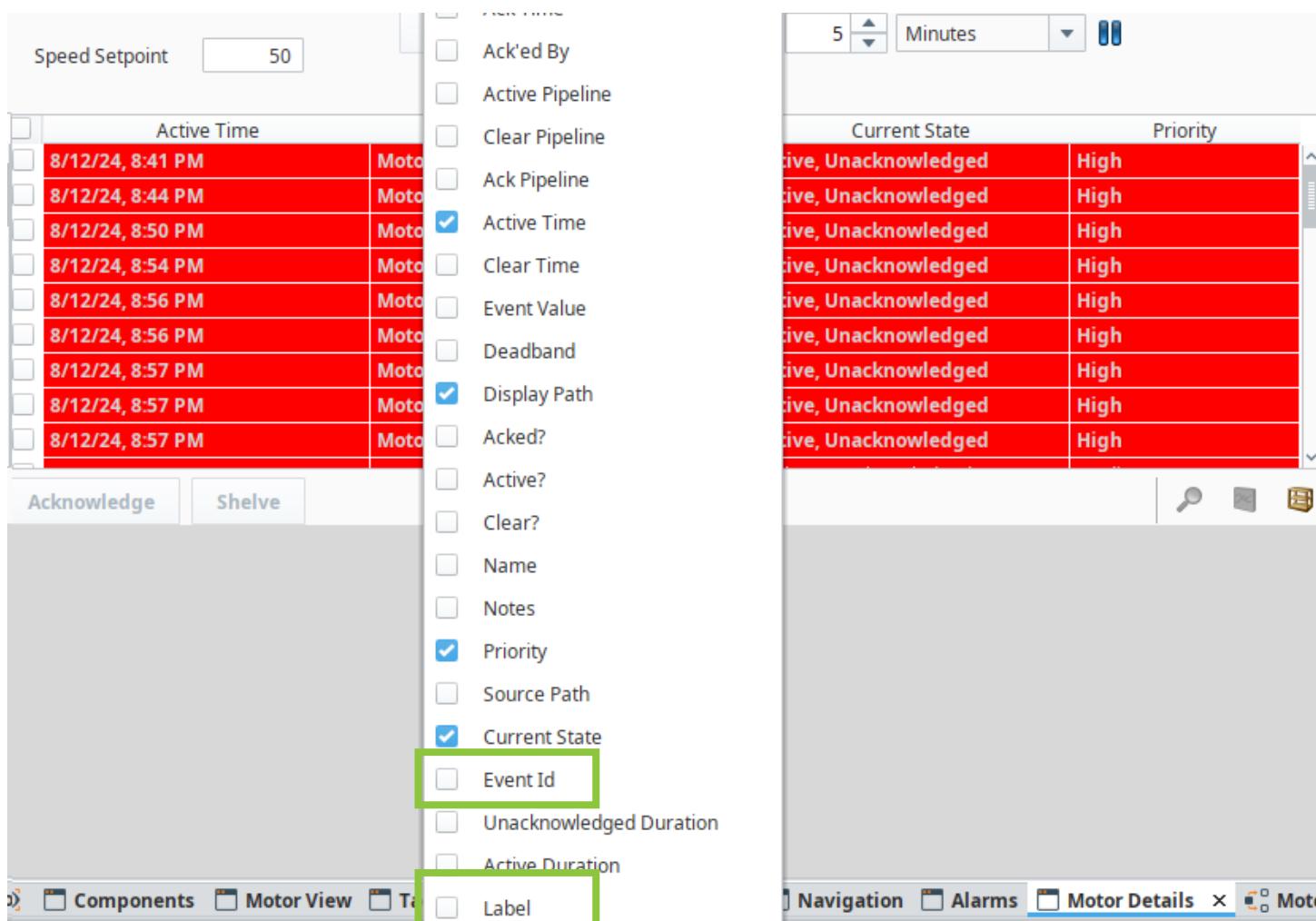
The last thing we'll add to our Popup window is a display that will show the live alarms for our specified motor. For this, we'll make use of the Alarm Status Table. See "Alarm Status Table" on page 267 to review this powerful component.

1. Expand the **Components Palette**.
2. Click in the Filter box.
3. Click  to clear to previous filter.
4. Enter **alarm**.
5. Drag an **Alarm Status Table** to the window.
6. Size and position the table to take up the bottom half of the window.

Customize the Columns

We can remove the columns from the table that we don't really need.

1. Click .
2. Right-click on any header cell in the table.
3. Click the checkboxes to remove **Event Id** and **Label**.
4. Resize the remaining columns to fit.



The screenshot shows the configuration and runtime view of an Alarm Status Table. On the left, the Components Palette is expanded, showing a list of alarm properties. The 'Current State' and 'Priority' checkboxes are checked. The 'Event Id' and 'Label' checkboxes are highlighted with green boxes, indicating they are being removed. On the right, the Alarm Status Table is displayed with 10 rows of data. The columns are 'Current State' and 'Priority'. All rows show 'Active, Unacknowledged' in the 'Current State' column and 'High' in the 'Priority' column.

Current State	Priority
Active, Unacknowledged	High

Filter the Table

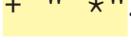
By default, the Alarm Status table will show all alarms on our system. We need to configure a filter that will only allow alarms for the current motor to be shown.

We explored **Display Path Filters** back in “Filter the Table” on page 270, and can use them again here to filter the table for the specific motor we want to show.

We want to configure a Display Path Filter that will only match alarms for the specified motor. The easiest approach is to filter out any alarms that don’t have the name of our Motor, which is something like Motor 1 or Motor 4, somewhere in the Display Path.

1. Click .
2. Click  for the **Display Path Filter** property.
3. Click **Expression**.

In order to filter for our motor number somewhere in the display path, we need an expression that looks for zero or more characters, followed by our motor name, followed by a space, and zero or more characters.

4. Enter  " *Motor " +
5. Click .
6. Click **MotorNumber**.
7. Enter  + " *".
8. Ensure that your expression looks like this:

```
"*Motor " + {Root Container.MotorNum} + " *"
```

Note: In this case, there are only 8 motors, so the space after the motor number isn't strictly necessary. However, if our operation expanded at some point and we got more than 10 motors, it would become critical, as the only way to differentiate between Motor 1 High Amps Alarm and Motor 10 High Amps Alarm is the space after the 1.

9. Click **OK**.

Test the Finished Popup

Our popup is complete, so let's return to the client and view the final result.

1. Click **File**.

2. Click **Save All**.
3. Click **Tools**.
4. Click **Launch project**.
5. Click **Launch (windowed)**.
6. **Log into** the client, using any user account.
7. Click **Motors**.
8. Click on any **motor template** instance.

Scripting

Scripting offers a significant degree of flexibility and customization where the standard available options fall short. There are two major scripting languages in Ignition: Python and Expression Language.

Let's now focus on Python in Ignition.

What Is Python?

Most of the time, when we talk about “scripting” in Ignition, we’re talking about Python scripting. Python is a general purpose programming language that was developed in the early 90’s and remains one of the most-widely used languages today.

Why do we use Python in Ignition? Python has a lot of advantages of other scripting languages, but the biggest one is simple: it gracefully interacts with Java. As Ignition is written in Java, we needed a scripting language that could easily interact with the underlying Java code, and Python does exactly that.

Python or Jython? Jython is an interpreter that converts Python to Java.

A Note on Versions

Python is currently available in version 3. However, in Ignition, we have to write Python 2.7. This is simply because Jython is still using 2.7. At some point, the folks behind Jython will update to 3, and then we’ll take a look at it and, if it makes sense, upgrade the version in Ignition. But for the foreseeable future, we will be using Python 2.7.

This is very important to understand because Python 3.0 (and they’re now well beyond that) represented a significant upgrade in the language. If you have any training in Python, it might be in Python 3. If you look online or get a book on the language, odds are it will be on version 3. So it’s important to remember that we are using a slightly outdated version of Python, and there are times when you will see coding techniques online that will not work in Ignition.

Python fun fact: while the logo for Python represents two coiled snakes, the language is actually named for Monty Python, the 70s-era British comedy group.

Getting Started in Python

Python is easy to learn because its syntax is intuitive and easy to read.

Script Console

We want to begin exploring Python by writing and testing some simple code.

To do this we need an environment to write our code in. To get started, will we use the Script Console in Designer.

1. Click **Tools**.

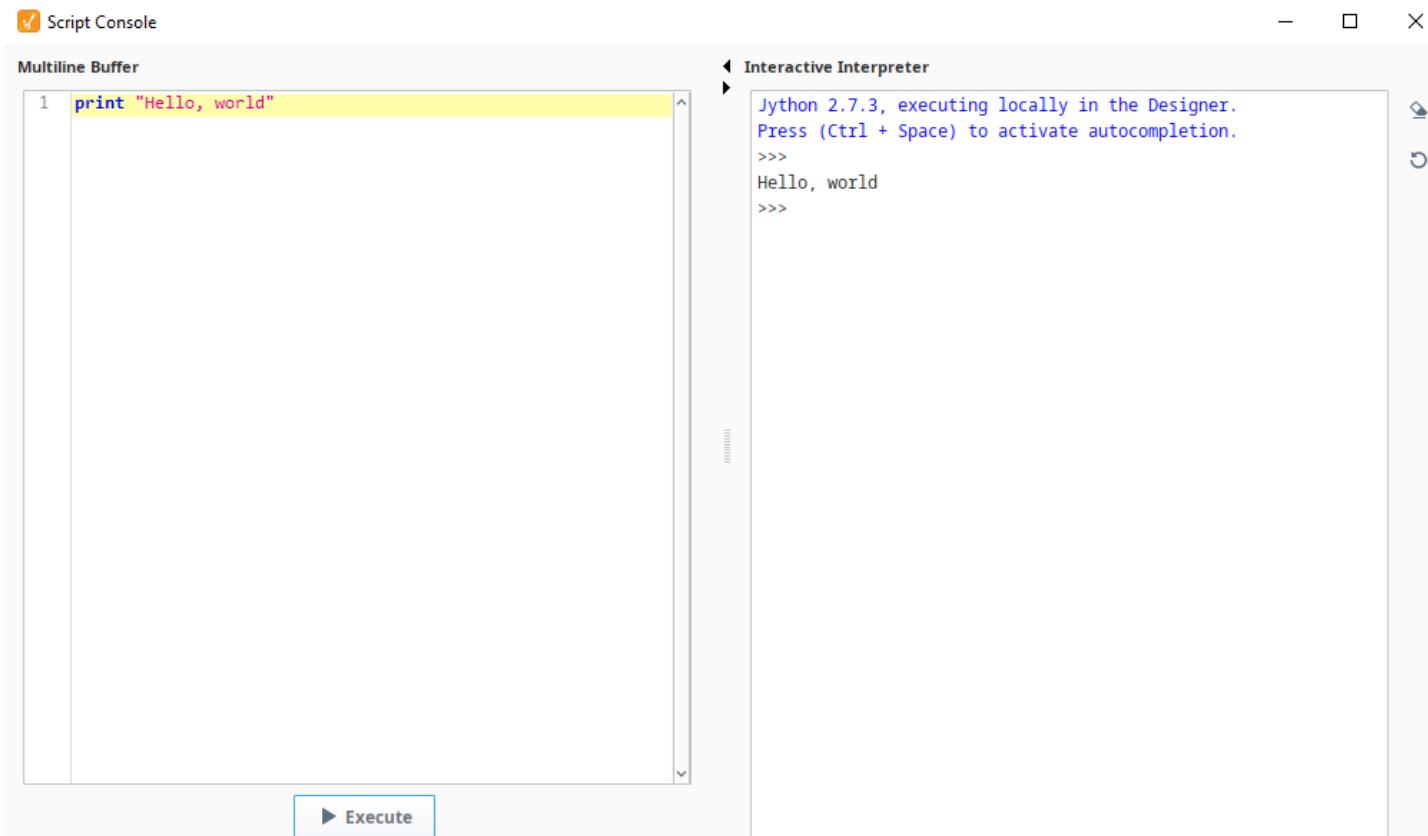
2. Click **Script Console**.

The Script Console is a simple interactive editor that allows us to write Python on the left and execute it. The result of the code will be displayed on the right.

Note: The Script Console does not permanently save code. It will only remember code that you have written in this session of Designer. In a bit, we will look at how you can write code that is saved with the project.

Let's start with an example that has been commonplace in beginning programming books since the 1970s: displaying the phrase "Hello, world" on the screen.

1. Type `print "Hello, world"`
2. Click **Execute**.



The screenshot shows the Script Console window with two main panes. The left pane, titled 'Multiline Buffer', contains a single line of Python code: `1 print "Hello, world"`. The right pane, titled 'Interactive Interpreter', shows the output of the code execution. It displays the Python version and path, followed by the output of the `print` statement: `>>> Hello, world`. At the bottom left of the Multiline Buffer pane is a blue 'Execute' button.

The `print` keyword is a handy tool in Python, allowing you to put text into the output console. This is useful for debugging your scripts.

Python Syntax

Now that we have written a line of code, we need to discuss Python's syntax. The syntax of a programming language is like the grammar of a spoken language. Following grammatical rules ensures that what we say makes sense to other people listening to us, and following syntax rules in programming makes sure that the tool that will run the code (in Python, like other scripting languages, that tool is called an interpreter) knows what it is supposed to do.

Basic Syntax Rules

Thankfully, Python has a syntax that makes it fairly easy to learn. The two most important rules to follow are:

Case sensitive Everything in Python is always case sensitive, meaning that whether or not you capitalize things always matters. Pay close attention to this as we move forward.

Whitespace sensitive Unlike many other languages, Python is whitespace sensitive. In particular, Python cares about when you press Enter in your code, and how you indent your code. You also need to pay close attention to this as we move forward.

Comments

Comments are notes that you can leave in your code. They can be very useful in describing what the code does, and why you chose to write the code in the way you did. Many times your comments will help you remember what the script does when you haven't gone through it in a long time.

Comments in Python begin with the # character. You might call that character a hash sign, or a pound sign, or a number sign, but regardless of what you call it, Python will always ignore anything on the line that follows it. The comment can be on its own line or be at the end of an existing line of code.

Note: Technically, Python does not support multi-line comments. You will find resources on line that include what appear to be multi-line comments in Python, but those are technically hacking a quirk in the way Python handles strings. Because they are not strictly supported by the language, we will not be using them in class.

Throughout Ignition, you can select one or more lines of code and use the keyboard shortcut Ctrl-/ to either add the comment character to the beginning of every selected line. If you select some code that is already commented, the same keyboard shortcut will remove the comment symbols.

Keywords

If syntax is the grammar of a programming language, keywords are its vocabulary. Thankfully, unlike spoken languages that have massive vocabularies (it's estimated that modern English has more than 5 million words), programming languages tend to stick to fairly small lists of keywords. We aren't going to list them here, because it's not important that you know all of them at this point. However, any coding environment within Ignition will make keywords bold and blue, and like everything else in the language, remember that they are case sensitive, so for example `True` is a keyword, but `true` is not.

Where Is Python Used?

Python is used in many places in Ignition. You can attach scripts to components, but you can also write code that becomes part of your project, or have code that directly runs on the Client or the Gateway.

Event Handlers

Event handling enables you to use scripting to respond to a wide variety of events that components fire. This lets you configure windows that are very interactive and are an important part of project design in Vision and Perspective.

An event can be many things, such as a mouse click, a key press, or simply a property change. Whenever these events occur, a script can be called to "handle" the event. Different components can fire different types of events.

Events are organized into event sets. For example, the mouse event set includes `mouseClicked`, `mousePressed`, and `mouseReleased`. Many times the events in an event set share the same properties for their event object.

Creating and Accessing Variables

Variables, a common staple of all programming languages, are a way to store a value that you use later using a name that is (hopefully) easy to remember. Even if you have no prior programming experience, you are likely familiar with the concept of a variable from middle school algebra: in the formula $x+1=2$, x is a variable.

Data Types

Python, like all programming languages, uses data types to differentiate between values. Python has many data types, but for our purposes in this class we really only need to care about four of them:

Strings A string is any sequence of characters, usually text. Words, sentences, or even individual letters are strings. Note that in certain cases, numbers can be represented as strings.

Integers An integer is a whole number, like 42.

Floats Floats are decimal numbers.

Booleans A Boolean value is literally either True or False.

Note: If you have a computer science degree, you might note that the above is a vast over-simplification. That is true, but it's really all we need to understand at this point in learning scripting.

Python is sometimes referred to as a “loosely typed” or “dynamically typed” language, which means that while every variable will have a value with a set data type, you do not need to declare the data type when you define the variable. Instead, Python will look at the value you assign and decide on the data type for you. Sometimes, that is fine, but other times it may lead to issues. We will explore one of those issues later.

Variable Names

Variable names must begin with a letter or an underscore character. Importantly, they cannot start with a number. After the first character, they can contain letters, underscores, and numbers.

Because variable names, like everything else in Python, are case sensitive, and also because they cannot contain spaces, you need to develop a consistent method of writing multi-word variable names.

For this class, we will be using a method known as camel case for all names. Camel case involves removing the spaces and capitalizing the first word of every word in the name, but leaving the very first letter lowercase.

In other words, if we want to represent “first name,” we would write it as firstName. If we wanted something like “is not admin”, we should use isNotAdmin.

There are other valid ways to represent multi-word variables in Python, but there’s a very important reason why we want to use camel case: every built-in name in Ignition

is written with camel case. So, if we use it for our names, we won't have to worry about how variable names are written when we're referring to both our own stuff and the stuff built into Ignition. It'll all be the same.

Create Variables

Let's create a few variables to see how they work.

1. Type `firstName =`.
2. Type your first name, enclosed in quotation marks.
3. Press **Enter**.
4. Type `lastName =`.
5. Type your last name, enclosed in quotation marks.
6. Type `print firstName, lastName.`
7. Click **Execute**.

The screenshot shows the Ignition Script Console interface. On the left, the 'Multiline Buffer' pane contains the following Python code:

```
1 firstName = "Mal"
2 lastName = "Reynolds"
3 print(firstName, lastName)
```

The third line, `print(firstName, lastName)`, is highlighted with a yellow background. At the bottom of this pane is a blue button labeled 'Execute'. On the right, the 'Interactive Interpreter' pane displays the output:

```
Jython 2.7.3, executing locally in the Designer.
Press (Ctrl + Space) to activate autocompletion.
>>>
Mal Reynolds
>>>
```

Your first and last name will appear on the right side of the console.

You will also see a nice feature of Python here: you can provide a comma-separated list of variables to the print statement, and it will output them all together.

Adding Code to Components

While the script console is a great place to test out a script, the downside as mentioned earlier is that it won't save our work if you close Designer. It also cannot interact with components on a window.

Because of these limitations, most of the time code will be written in other places in our project. And one particularly common place is on a component in a window.

Create a Script Window

To begin, let's create a window that we can use to test out a variety of scripts.

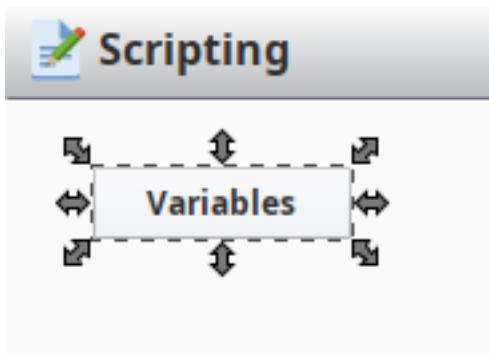
1. Click **Vision**.
2. Expand **Windows**.
3. Expand **Main Windows**.
4. Right-click **Empty**.
5. Click **Duplicate**.
6. Right-click **Empty (1)**.
7. Click **Rename**.
8. Type **Scripting**.
9. Press **Enter**.
10. Double-click the **header text**.
11. Type **Scripting**.

Add a Button

The easiest component to add a script to is the Button. In fact, the Button component does not do anything on its own—you must write a script of some kind to get a Button to work.

12. Expand the **Components Palette**.
13. Click in the **Filter** box.

14. Click  to clear to previous filter.
15. Type **button**.
16. Drag a **Button** to the Window.
17. Click the **Text** property.
18. Enter **Variables**.



Because you have to add some kind of script to get a Button to do anything, double-clicking the Button opens the Component Scripting dialog box.

Component Scripting Dialog

We saw the Component Scripting Dialog box in the “Opening a Popup” on page 327, but we only used the first tab—Navigation.

The actionPerformed Event

We have discussed events briefly before. Buttons will almost always use the `actionPerformed` event, which is triggered regardless of how the button is activated. In fact, this is so common that this event is pre-selected for you when you open the Component Scripting dialog from a button.

Other Tabs

While we will not be using any of the other tabs except the last one, it’s worth taking a moment to discuss what each one does.

Navigation The Navigation tab allows us to navigate somewhere—most often, another window—when the button is clicked.

Set Tag Value You can use this tab to set a specific Tag to a given value when the button is clicked.

SQL Update Trigger a query to the database to update values on clicking the button.

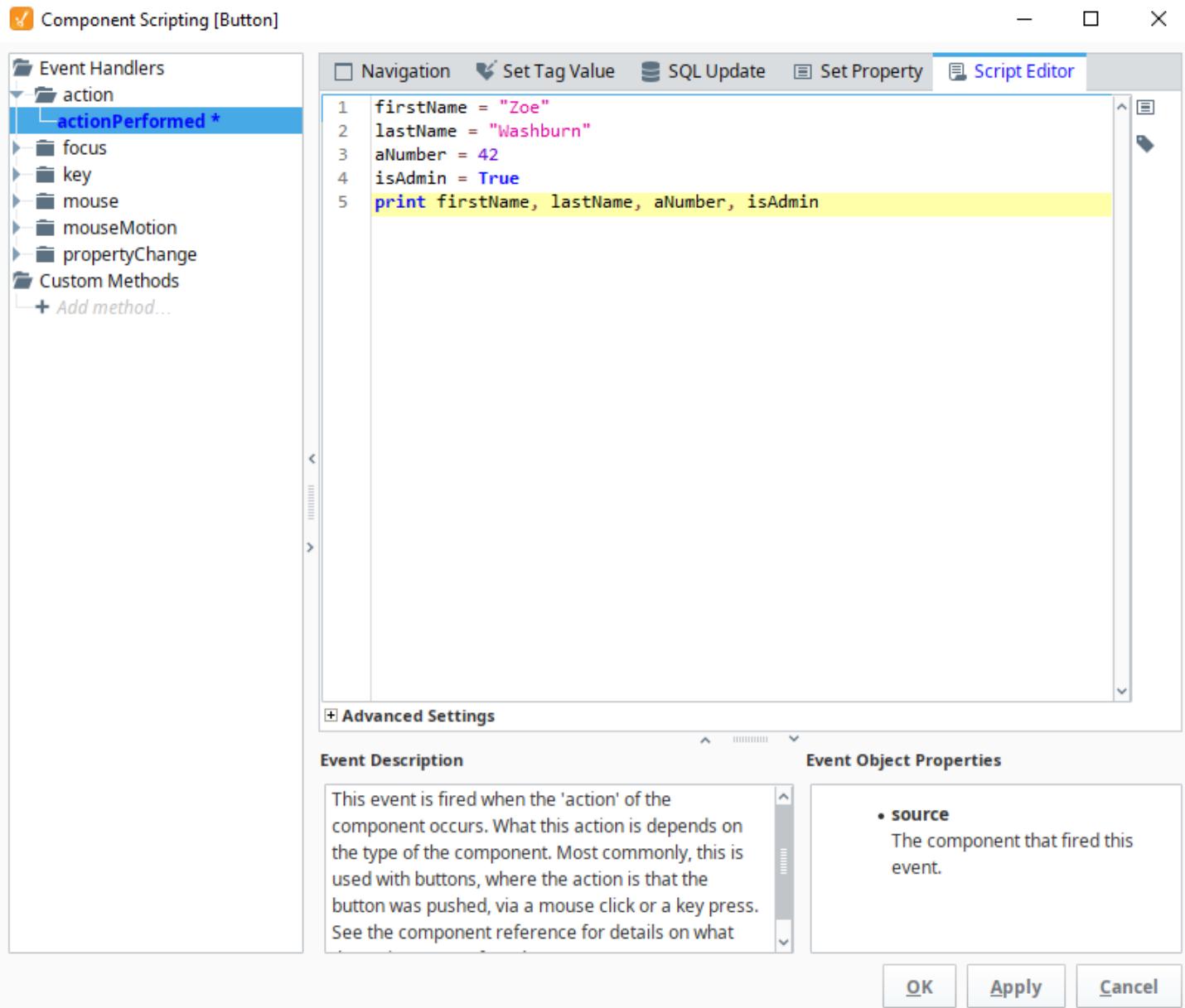
Set Property Change the property of a component to a different value when clicking the button.

Script Editor Write your own code. This is the tab we will use for the remainder of this section of the class.

Add Variables to the Script

Now that we have an understanding of the dialog box, let's revisit variables. First, we'll recreate the basic variables we had in our Script Console.

1. Double-click the **Button**.
2. Click **Script Editor**.
3. Enter `firstName =`
4. Enter your first name, enclosed in quotation marks.
5. Press **Enter**.
6. Enter `lastName =`
7. Enter your last name, enclosed in quotation marks.
8. Press **Enter**.
9. Enter `aNumber = 42`
10. Press **Enter**.
11. Enter `isAdmin = True`
12. Press **Enter**.
13. Enter `print firstName, lastName, aNumber, isAdmin`



Testing Code on a Button

In order to execute the code attached to a Button to test it, we first need to save the information in the dialog box. The normal way of doing this is to click OK, which you should know from all of your other computer use saves the changes made in the dialog and closes it. However, we also have an Apply button. Clicking that saves the changes in the dialog but leaves it open. This is particularly helpful because the Component Scripting dialog is what is known as a non-modal dialog, which is just fancy computerspeak for “a dialog box that can stay open because we can still interact with the rest of the program.”

1. Click **Apply**.

2. Move the **Component Scripting** dialog box so that you can see the rest of Designer.
3. Click ► .
4. Click the **Button**.

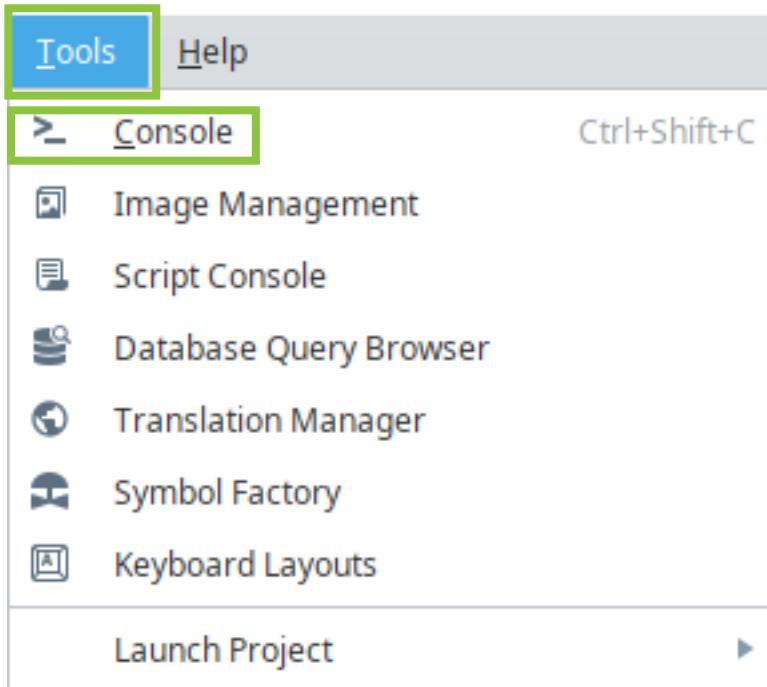
Clicking the Button executes its code ... but where did it go?

When we used the print statement in the Script Console, the results showed up in the console. But when we try to use it on a component on a window, we need to open a console for the statement to write to.

The Console Panel

We have a Console Panel in Designer that, among other things, can display the results of executing a print statement. Even though it's opened from the Tools menu, it acts like any other panel, so we have the ability to dock it in place.

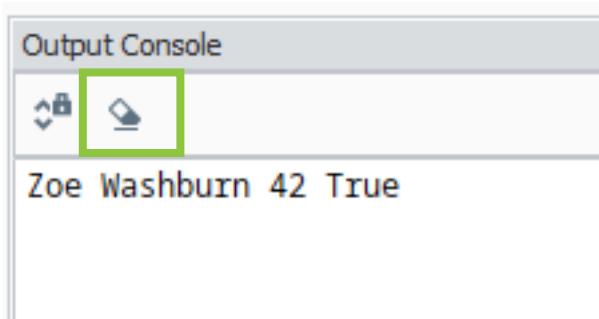
1. Click **Tools**.
2. Click **Console**.



3. Press and hold your mouse on the **Console's title bar**.
4. Drag the console so that it **docks at the bottom** of the screen.

The console does not only display print statements. Every time Designer starts, a lot of information is printed to the console, which is what you see in it now. The console will also display errors if they occur. So that we can focus in on our print statement, let's clear the console, then run our code again.

5. Click **Clear**.
6. Click the **Button**.



This time, you will see the result of the print statement in the console.

Basic Math

While advanced math requires a bit more work, basic mathematical operations can be executed directly in Python.

Python supports a fairly standard set of mathematical operators:

- + Addition
- Subtraction
- * Multiplication
- / Division

% Modulo. This might require a small bit of explanation. The modulo is the remainder of a division problem. Think back to when you first learned division: you learned that 5 divided by 2 is 2 with a remainder 1. The modulo operator simply solves a division problem, but rather than returning the result as a decimal, as the division operator does, it just returns that remainder. So, $5 \% 2$ would return 1. You may be surprised at just how often this is useful in programming. In fact, we will see an example of its use before we finish this chapter.

Let's take a quick look at doing math in Python.

1. Return to the **Component Scripting** dialog.

2. Select **all of the code**.

Note: The keyboard shortcut Ctrl-a will select all code.

3. Press **Ctrl-/** to add comment characters to the beginning of every line.

4. Click to **deselect** the code.

5. Press **Enter** after the last line.

6. Enter **add = 10+32**

7. Press **Enter**.

8. Enter **subtract = 52-10**

9. Press **Enter**.

10. Enter **multiply = 7 * 6**

11. Press **Enter**.

12. Enter **divide = 420/10**

13. Press **Enter**.

14. Enter **modulo = 8814%43**

15. Press **Enter**.

16. Enter **print add, subtract, multiply, divide, modulo**

17. Click **Apply**.

18. If necessary, **move** the Component Scripting dialog.

19. Click **Clear**.

20. Click the **Button**.

The answer to all of those problems is, of course, 42.

Conditional Processing

Programming is, ultimately, just about getting computers to do what we want. And that will inevitably require decision making.

In Python, you can create an if block to perform basic binary logic; that is, evaluate a value and do something if that evaluation is true, and something else (or possibly nothing) if it is false.

Whitespace and Code Blocks

This is where Python's use of whitespace becomes critical. So far, we have only used code statements, which are single lines. But now, we need more than one line to express our code: at a minimum, the if statement needs to be on a line, and the code to execute if the statement is true on a second line. But in order for Python to know that the second line is related to the first, that second line must be indented.

It is unimportant how much indentation you use, and also, it does not matter if you indent using the tab key or by pressing the spacebar one or more times. All that matters is consistency: if you use tab to indent in one place, you need to use it to indent the same amount for every other related line.

In addition, that first line—the one with the actual if statement—needs to end with a colon.

Comparison Operators

Just as Python uses a set of symbols to do math, it uses a set of symbols to perform comparisons. All of these comparisons will result in either True or False.

`==` Two equal signs means “equal to.” Note that we need two equal signs here—one equal sign, as we have already seen, assigns a value to a variable.

`>` Greater than

`<` Less than

`>=` Greater than or equal to

`<=` Less than or equal to

`!=` Not equal to

Writing a Basic If Statement

Let's create a new button, so that we can keep the code we have already written for future reference. However, rather than getting a brand new button, we can simply duplicate the button we already have, and then replace the code.

1. Click **OK** to close the existing Component Scripting dialog.

2. Click .
3. Click the **Button**.
4. Press and hold **Ctrl** on your keyboard.
5. **Drag** the Button down.
6. Release **Ctrl**.
7. Double-click the value of the Button's **Text** property.
8. Enter **If**.
9. Press **Enter**.
10. Double-click the **Button**.
11. Click in the code.
12. Press **Ctrl+A** on your keyboard.
13. Press **Delete**.
14. Enter `var = 42`
15. Press **Enter**.
16. Enter `if var < 42:`
17. Enter `print "var is less than the answer."`
18. Ensure your code looks like this:

```
var = 42  
if(var < 42):  
    print "var is less than the answer."
```

19. Click **Apply**.
20. Move the **Component Scripting** dialog out of the way.
21. Click .
22. Click .
23. Click the **Button**.

Nothing should be output to the console, because the condition is false, so print never executes.

Else

In this case, we have a single test. If that test is true, the print statement executes. If it's false, nothing happens.

But what if we do want something to happen? In that case, you can add an else statement.

Pay close attention to whitespace. The else statement needs to be at the same level of indentation as if, and the statement to execute on else needs to be indented the same amount as the statement to execute when the if statement is true.

1. Return to the **Component Scripting** dialog.
2. Enter **else:** as the last line of code.
3. Press **Enter**.
4. Enter **print "var is greater than the answer"**
5. Ensure your code looks like this:

```
var = 42
if(var < 42):
    print "var is less than the answer."
else:
    print "var is greater than the answer."
```

6. Click **Apply**.
7. Move the **Component Scripting** dialog out of the way.
8. Click .
9. Click the **Button**.

This time we get something to print because of the else allowing for some logic on when the condition is not met.

Elif

Lastly, you can use elif to add multiple conditions. Short for “else if”, elif allows you to check another condition when the previous one is not met. This form can optionally have a catch-all else clause at the end.

1. Return to the **Component Scripting** dialog.

2. Enter `elif (var == 42):` on line 4.

3. Press **Enter**.

4. Enter `print "var is the answer"`

5. Ensure your code looks like this:

```
var = 42
if(var < 42):
    print "var is less than the answer."
elif(var == 42):
    print "var is the answer."
else:
    print "var is greater than the answer."
```

6. Click **Apply**.

7. Move the **Component Scripting** dialog out of the way.

8. Click .

9. Click the **Button**.

Try changing the value of `var` to see how it impacts which condition is true.

Lists

Lists are one of the most important concepts to understand, not just for Python in general but for using Python in Ignition.

Whereas variables store a single piece of data, lists allow us to store multiple data points in a single object that can be then be referenced as a whole, but also still referenced as individual pieces as well.

To examine lists, we will create a new button and add code to it.

1. Click **OK** to close the existing Component Scripting dialog.

2. Click .

3. Click the **Button**.

4. Press and hold **Ctrl** on your keyboard.

5. **Drag** the Button down.

6. Release **Ctrl**.
7. Double-click the value of the Button's **Text** property.
8. Enter **Lists**.
9. Press **Enter**.
10. Double-click the **Button**.
11. Click in the code.
12. Press **Ctrl+A** on your keyboard.
13. Press **Delete**.
14. Enter `nums = range(5)`
15. Press **Enter**.
16. Enter `print nums`
17. Click **Apply**.
18. Move the **Component Scripting** dialog out of the way.
19. Click .
20. Click the **Button**.

The range function in Python creates a list of integers. The lists starts at zero, and contains the number of integers you specify inside the parentheses; in this case, we get a total of five items in the list, ranging from 0-4.

Note how the output looks: `[0, 1, 2, 3, 4]`

Of particular importance are the square brackets. These are used whenever we are creating a list.

You can also use a list that you create.

1. Return to the **Component Scripting** dialog.
2. Enter `#` at the beginning of the print statement to comment it out.
3. Enter `ships = ["Enterprise", "Galactica", "Serenity"]`
4. Enter `print ships`
5. Ensure your code looks like this:

```
nums = range(5)
```

```
#print nums
```

```
ships = ["Enterprise", "Galactica", "Serenity"]
```

```
print ships
```

6. Click **Apply**.
7. Move the **Component Scripting** dialog out of the way.
8. Click .
9. Click the **Button**.

The list will appear in the Console window. Note that Ignition replaces the double quotes with single quotes. Python accepts either, so this does not matter; when you create the list, you can as easily use single quotes.

Referencing Items in a List

You can reference an item in a list by using its index number. Python uses zero-indexing, meaning that the first item in the list is at index 0, and second at index 1, and so forth.

1. Return to the **Component Scripting** dialog.
2. Enter **#** at the beginning of the print statement to comment it out.
3. Enter **print ships[1]**
4. Click **Apply**.
5. Move the **Component Scripting** dialog out of the way.
6. Click .
7. Click the **Button**.

You will see that the second item in the list is printed.

Adding to the List

You can add items to a list with the `append()` function. This function is called on the list using dot notation: you type the name of the list, then a period (dot), and then the function. We will see many more uses of dot notation as we get more advanced in Python.

1. Return to the **Component Scripting** dialog.
2. Enter **#** at the beginning of the print statement to comment it out.
3. Enter **ships.append("Falcon")**

4. Enter `print ships`
5. Click **Apply**.
6. Move the **Component Scripting** dialog out of the way.
7. Click .
8. Click the **Button**.

Lists With Multiple Types

Python lists can contain items of different data types. They can even contain other lists.

1. Return to the **Component Scripting** dialog.
2. Enter `#` at the beginning of the print statement to comment it out.
3. Enter `stuff = [1, "banana", 3.14, ships]`
4. Enter `print stuff`
5. Click **Apply**.
6. Move the **Component Scripting** dialog out of the way.
7. Click .
8. Click the **Button**.

You will see that the new list contains a integer, a string, a decimal, and the other list.

You can reference an item in a nested list by providing the index of the nested list, then the index of the item in that list.

1. Return to the **Component Scripting** dialog.
2. Enter `#` at the beginning of the print statement to comment it out.
3. Enter `print stuff[3][2]`
4. Click **Apply**.
5. Move the **Component Scripting** dialog out of the way.
6. Click .
7. Click the **Button**.

Loops

Like any other programming language, Python allows us to execute one or more lines of code repeatedly with a loop. Unlike other languages, though, Python really only contains two types of loops, and in Ignition, we primarily only rely on one.

For Loop

The for loop in Python has one purpose: looping over a list. Because of this, we will continue to work with the Lists button from the prior section.

1. Return to the **Component Scripting** dialog.
2. Enter `#` at the beginning of the print statement to comment it out.
3. Enter `for num in nums:`

This line initializes the for loop. We are creating a new variable, num, which will represent each item as the loop iterates.

Note the colon at the end of the line. We are going to create a code block here, and the colon is part of what designates that what follows will be a block, just like we did with our if code blocks. And, just like the if, the code that we want to execute within the for loop must be indented.

4. Press **Enter**.
5. Enter `print num`
6. Click **Apply**.
7. Move the **Component Scripting** dialog out of the way.
8. Click .
9. Click the **Button**.

You will see each number from the numbers list print on its own line, because each number is executing in a separate print statement.

While this example is simply printing the items in a list with a for loop, you can in practice do anything you want to the items in the loop. We will see examples of this later.

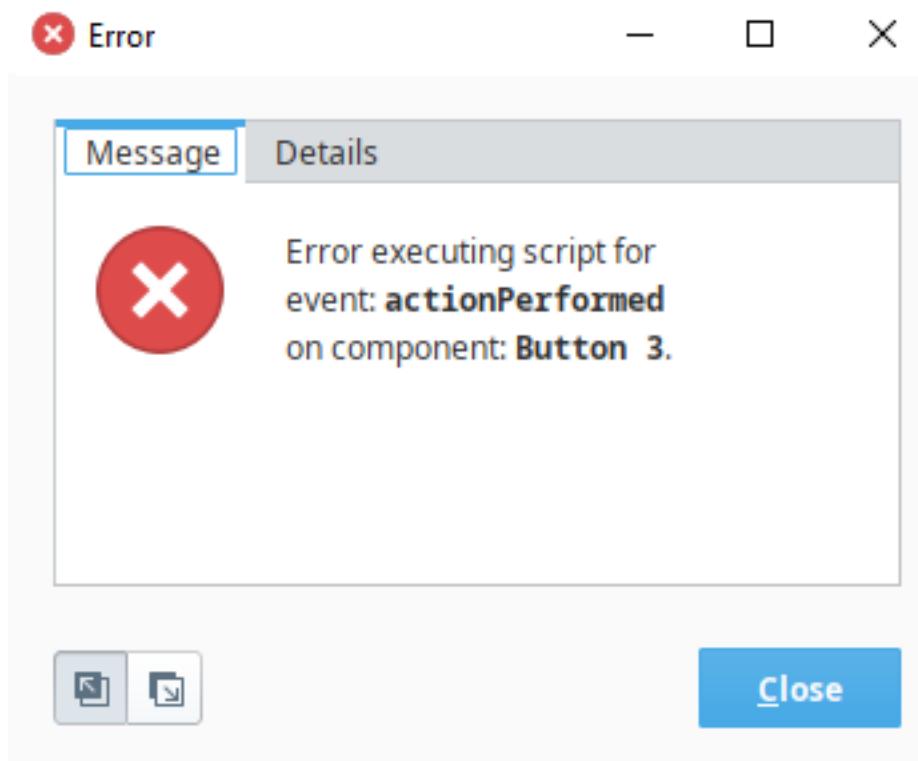
String Formatting

Often, you may need to take multiple strings and combine them, or push a variable into a string that will later be displayed to the user. There are a few different ways of doing

this: string concatenation and string formatting. Concatenation is basically a fancy word for “add”, so string concatenation adds several strings together using the plus symbol. String formatting, on the other hand, allows you to enter values into specific points into the string.

1. Click **OK** to close the existing Component Scripting dialog.
2. Click .
3. Click the **Button**.
4. Press and hold **Ctrl** on your keyboard.
5. **Drag** the Button down.
6. Release **Ctrl**.
7. Double-click the value of the Button’s **Text** property.
8. Enter **Strings**.
9. Press **Enter**.
10. Double-click the **Button**.
11. Click in the code.
12. Press **Ctrl+A** on your keyboard.
13. Press **Delete**.
14. Enter `temp = 75`
15. Press **Enter**.
16. Enter `city = "Folsom"`
17. Enter `print "The temperature in " + city + " is currently " + temp + " degrees."`
18. Click **Apply**.
19. Move the **Component Scripting** dialog out of the way.
20. Click .
21. Click .
22. Click the **Button**.

You will receive the follow error.



This error is occurring because we are trying to combine two datatypes in a single statement: we have a series of strings, and an integer.

Type Casting

There are two solutions to this issue.

First, we can use type casting, which means forcing Python to convert a variable from one type to another. This is possible in this situation, but an integer and a string are compatible; in other words, it's possible to treat an integer as a string. In fact, almost any type can be cast to a string. The opposite, however, is not always possible: while you can write 42 as a string, you cannot write platypus as a number.

Type casting is performed by using a set of functions in Python. Each of the basic data types has a corresponding type function.

Let's go ahead and fix that first example using type casting.

1. Return to the **Component Scripting** dialog.
2. Enter `str(` before `temp` on line 3.
3. Enter a **closing parenthesis** after `temp`
4. Ensure your code looks like this:

```
temp = 93
city = "Rocklin"
print "The temperature in " + city + " is currently " + str(temp) + " degrees."
```

5. Click **Apply**.
6. Move the **Component Scripting** dialog out of the way.
7. Click .
8. Click the **Button**.

You will now see the statement print correctly, without error.

Python String Formatting

Type casting exists in almost every programming language, but Python has a special way of formatting strings that is unique to the language.

This technique allows us to input special placeholders in a string, and then follow the string with a special function that allows you to replace the placeholders with formatted variables.

1. Return to the **Component Scripting** window.
2. Enter `print "The temperature in %s is currently %i degrees."%(city, temp)`

The first placeholder, %s, designates that the value we will put in its place will be a string. The second placeholder, %i, will need to be replaced by an integer. Then, following the string, we put another percent symbol, and in a pair of parentheses, we put the values we want to use to replace the placeholders. Note that the order matters: the list of values replaces the placeholders in order, left to right.

3. Click **Apply**.
4. Move the **Component Scripting** dialog out of the way.
5. Click .
6. Click the **Button**.

When you run the code this time, you will see two identical sentences, showing that the more traditional programming method of using type casting and the Pythonic method of string formatting give you the same result.

Both print statements will display the same sentence in the Console using different methods to create them.

Note: String formatting is one of the things that changed most significantly from Python 2 to Python 3. It's important to remember that in Ignition, we are using Python 2.7. So, if you search online for help with string formatting, you are likely to get a lot of results that use newer techniques that will not work in Ignition. A useful resource is pyformat.info, which is a website that not only shows all of the possibilities for string formatting, but also designates which techniques work in the “old style” (i.e., Python 2, and thus Ignition) and which ones are new to Python 3. Note that Inductive Automation is not affiliated with this website in any way and is only providing the link as an additional resource.

Python in Ignition

So far, we have only looked at generic Python. Everything we have done so far could be done in any code editor and executed against a stand-alone installation of Python. Now, we want to look at how we can write Python code that is specific to Ignition.

System Functions

The Ignition scripting API, which is available under the module name “system”, is full of functions that are useful when designing projects in Ignition. These functions are not part of the standard Python language, but are instead built into Ignition to ease the execution of certain common tasks and interact with various systems in Ignition. In the example below, we cover some of the things that you can do with system functions.

1. Click **OK** to close the existing Component Scripting dialog.
2. Click .
3. Click the **Button**.
4. Press and hold **Ctrl** on your keyboard.
5. **Drag** the Button down.
6. Release **Ctrl**.
7. Double-click the value of the Button’s **Text** property.
8. Enter **System**.
9. Press **Enter**.

10. Double-click the **Button**.
11. Click in the code.
12. Press **Ctrl+A** on your keyboard.
13. Press **Delete**.
14. Enter **system**. Be sure to type the period.

Now that we are working with Ignition-specific code, we will start to see code hints. These will help us write code by typing less, which also allows us to avoid typos.

15. Enter **g** immediately after the period.
16. Press **Enter**.

Because **gui** was selected (as the only item) on the code hints, the script completed the rest for us, including the period we will need for the next step.

17. Enter **m** immediately after the period.
18. Press **Enter**.

Again, typing just the first letter was enough to select the item we need.

The **messageBox** function opens a modal message box on the screen, which is a fancy way of saying a dialog box that we must deal with before we can do anything else. The function takes two parameters: a message, and a title. Title is optional, so we are going to delete it.

19. Replace **message** with **"Hello, world."**
20. **Delete** **, title**
21. Ensure your code looks like this:

```
system.gui.messageBox("Hello world")
```

22. Click **Apply**.
23. Move the **Component Scripting** dialog out of the way.
24. Click .
25. Click .
26. Click the **Button**.

The message box will appear. You will need to click OK to close it.

There is also a lot of information we can get from these system functions.

1. Return to the **Component Scripting** dialog.
2. Enter # at the beginning of the **messageBox** statement to comment it out.
3. Enter `print system.util.getGatewayAddress()` You can use the code hints to save some keystrokes.
4. Be sure your code looks like this:

```
#system.gui.messageBox("Hello world")
```

```
print system.util.getGatewayAddress()
```

5. Click **Apply**.
6. Move the **Component Scripting** dialog out of the way.
7. Click .
8. Click the **Button**.

You should see your Gateway address, `http://localhost:8088`, printed in the Console.

9. Return to the **Component Scripting** dialog.
10. Enter # at the beginning of line 2 to comment it out.
11. Enter `print system.user.getUsers()`
12. Enter "default" to replace the userSource placeholder between the parentheses.
13. Be sure your code looks like this:

```
#system.gui.messageBox("Hello world")
```

```
#print system.util.getGatewayAddress()
```

```
print system.user.getUsers(userSource)
```

14. Click **Apply**.
15. Move the **Component Scripting** dialog out of the way.
16. Click .
17. Click the **Button**.

You should see something similar to this in the Console:

```
[User[username=admin, firstname=null, lastname=null, roles=[Administrator]],  
User[username=guest, firstname=null, lastname=null, roles=[]], User[username=oper,  
firstname=null, lastname=null, roles=[Operator]]]
```

Ask yourself: what does this look like in Python?

Hopefully, you said, “a list of lists.” And you would be right.

Of course, an important aspect of most Ignition projects is working with Tags, and we do have ways we can access Tags in scripting.

1. Return to the **Component Scripting** dialog.
2. Enter `#` at the beginning of line 3 to comment it out.
3. Enter `tags = system.tag.readBlocking()`

The required parameter for the `readBlocking` function is a list of Tag paths. (You can see this from the black tooltip that appears when you first type this code.) So, to specify the Tags we want, we first need a list.

4. Enter `[]` to replace the `TagPaths` placeholder between the parentheses.

Unfortunately, the code hints don’t state this, but each Tag path within the list needs to be passed as a string.

5. Enter `""` inside the square brackets.

We could enter the paths to the Tags we want, but it’d be easier (and less prone to error) if we instead copied them from the Tag Browser.

6. Expand **GenSim** in the Tag Browser.
7. Expand **Ramps**.
8. Right-click **Ramp0**.
9. Click **Copy Path**.
10. Return to the **Component Scripting** dialog.
11. Right-click **between the quotes** you entered in step 48.
12. Click **Paste**.

Let’s take advanTage of the function accepting a list of Tags and add a second one.

13. Enter `, ""` after the closing quotation mark.
14. Expand **DairySim** in the Tag Browser.
15. Expand **Overview**.
16. Expand **Motor 1**.
17. Right-click **Amps**.

18. Click **Copy Path**.
19. Return to the **Component Scripting** dialog.
20. Right-click **between the quotes** you entered in step 56.
21. Click **Paste**.
22. Click at the end of the line.
23. Press **Enter**.
24. Enter **print tags**
25. Be sure your code looks like this:

```
#system.gui.messageBox("Hello world")
#print system.util.getGatewayAddress()
#print system.user.getUsers("default")
tags = system.tag.readBlocking(["[default]_GenSim_/Ramp/
Ramp0","[default]_DairySim_/Overview/Motor 1/Amps"])
print tags
```

Note: The path should be written on one line.

26. Click **Apply**.
27. Move the **Component Scripting** dialog out of the way.
28. Click .
29. Click the **Button**.

You should see results similar to this:

**[[7.5439, Good, Mon Jul 29 13:08:34 PDT 2024 (1722283714688)],
[98.7552879530954, Good, Mon Jul 29 13:08:33 PDT 2024 (1722283713675)]]**

Again, ask yourself: what does that look like?

Once again, “a list of lists.” But, whereas **getUsers** did in fact return a list of lists, this result is a bit trickier.

Let’s try to use list syntax to get the first value out of that result—the value of Ramp0.

1. Return to the **Component Scripting** dialog.
2. Add **[0][0]** to the last line of code.

3. Ensure your code looks like this:

```
#system.gui.messageBox("Hello world")
```

```
#print system.util.getGatewayAddress()
```

```
#print system.user.getUsers("default")
```

```
tags = system.tag.readBlocking(["[default]_GenSim_/Ramp/  
Ramp0","[default]_DairySim_/Overview/Motor 1/Amps"])
```

```
print tags[0][0]
```

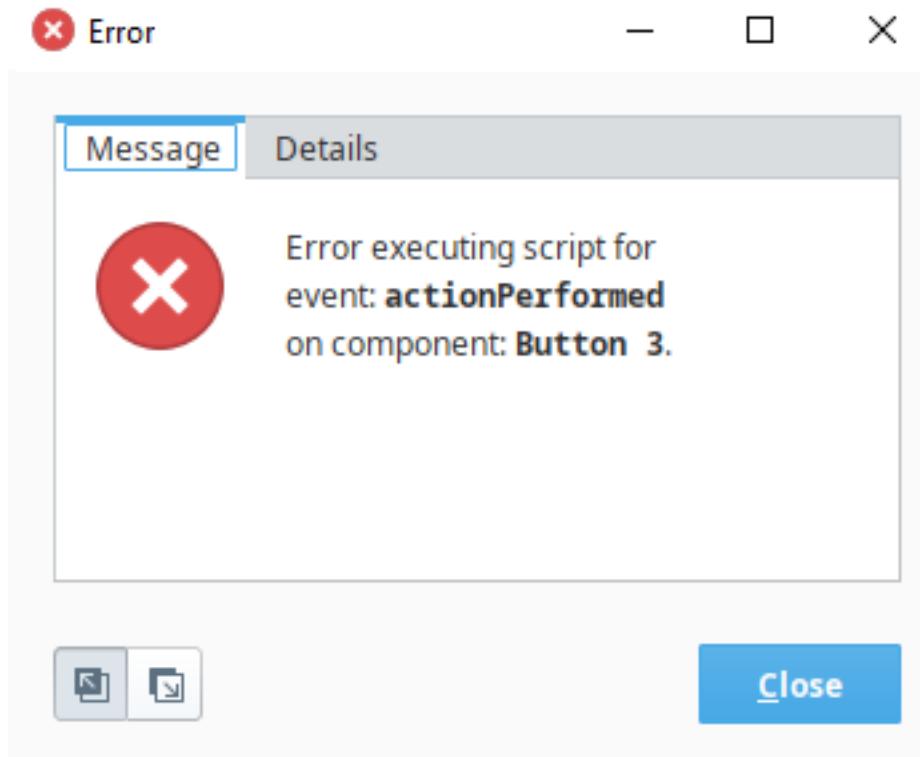
4. Click **Apply**.

5. Move the **Component Scripting** dialog out of the way.

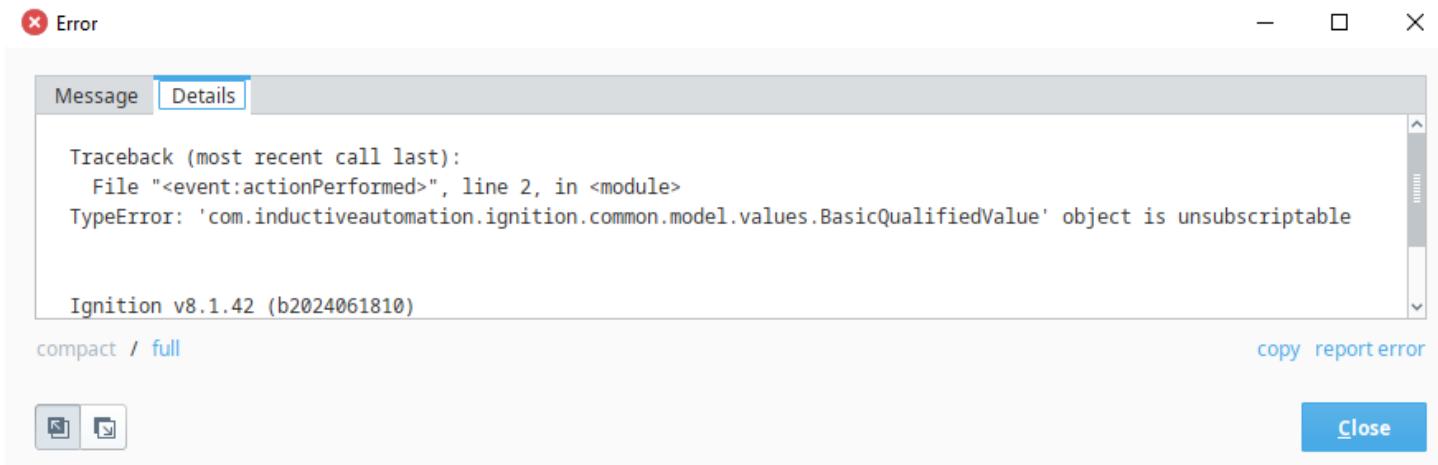
6. Click .

7. Click the **Button**.

You will get an error. Let's examine the error to see what's wrong.



8. Click **Details** on the error dialog box.
9. If necessary, expand the **width of the dialog** to see the whole message.



The error message should say, **TypeError: 'com.inductiveautomation.ignition.common.model.values.BasicQualifiedValue' object is unsubscriptable**.

What does that mean? We could just tell you, but you're likely to encounter many errors as you learn Python in Ignition, so it's worth exploring our on line documentation to find the answer.

Using the Documentation

As we've now seen, many times we will try to do something that looks right, but results in an error. Thankfully, the system functions in Ignition are all well documented.

1. Open the **Gateway web page** in a browser.
2. Click **Home**.
3. Click **Open Appendix**.



Appendix

A complete reference for components, expressions, and scripting functions in Ignition.

[Open Appendix](#)

The documentation appendix provides a complete reference to Ignition components, Expression Functions, System Functions, and Reference Pages, as well as a link to documentation for older versions of Ignition.

We know that we are working with system functions, so the answer to fixing that error can be found by following links in that section.

4. Click **System Functions**.
5. Click **system.tag**.
6. Click **readBlocking**.
7. Scroll to the **Returns** section.

We can see here that the **readBlocking** function does not in fact return a list of lists, but rather, a list of something called **QualifiedValue** objects. What is that? Well, thankfully, there's a cross reference right on that page.

8. Click **Scripting Object Reference**.

This page, which will automatically scroll to the section that explains a **QualifiedValue** object, shows us that even though this object looks like a list, it is in fact a special thing within Ignition. We can also see that it has three properties: value, quality, and timestamp. If you go back to Designer and look at the original output from the

readBlocking Tag, each of the two Tags is returning three things: the Tag's value, its quality ("Good"), and a timestamp of when the function read the Tag.

This tells us what we need to know in order to output that value: it's not the zero index of the list (because it isn't a list); instead, we can use value.

9. Return to **Designer**.
10. Return to the **Component Scripting** dialog.
11. **Delete** the last [0]
12. Enter .value Be sure to type the period.
13. Ensure your code looks like this:

```
tags = system.tag.readBlocking(["[default]GenSim/Ramp/Ramp0",
"[default]DairySim/Overview/Motor 1/Amps"])
```

```
print tags[0].value
```

14. Click **Apply**.
15. Move the **Component Scripting** dialog out of the way.
16. Click .
17. Click the **Button**.

There are certain aspects of code that you will memorize through repeated use, but the vast majority will be things you need to look up. Remember that our documentation is there to answer many of those questions. You can also use the user forums—there's also a link on the Gateway home page to those—to ask for help if you can't find the answer you need in the documentation.

Reading Component Values

So far, all of our scripts have been attached to buttons, and have relied on the code within the button to do what we need. However, it is of course entirely possible to write code that takes information from one component and uses it in another component.

Let's start with a fairly simple example: we will add a text field and a button. Our user will be able to type something into the text field, and then click the button to have whatever was typed appear in a message box.

1. Click **OK** to close the existing Component Scripting dialog.
2. Click .

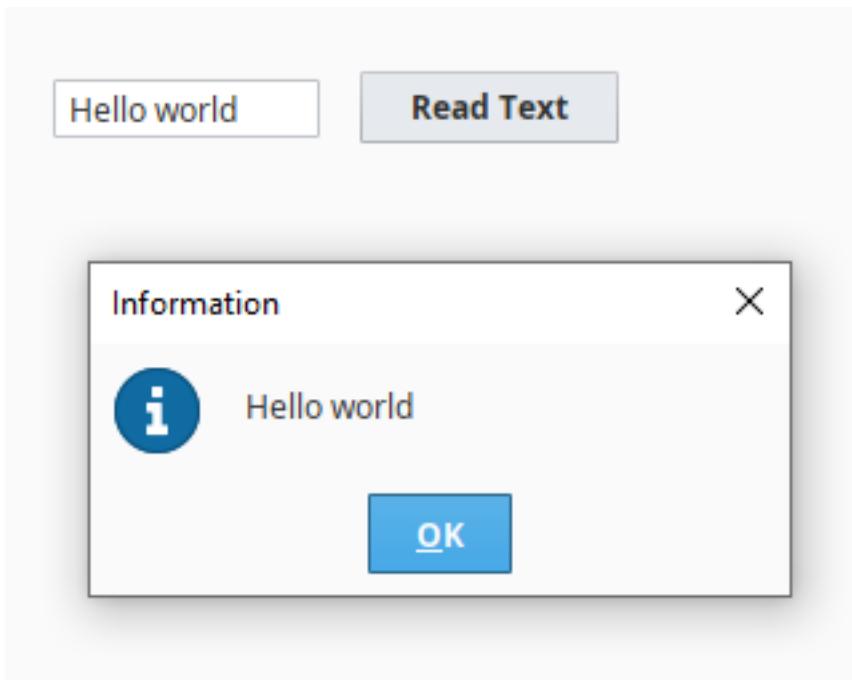
3. Expand the **Components Palette**.
4. Drag a Text Field component to the window.
5. Click a **Button**.
6. Press and hold **Ctrl** on your keyboard.
7. **Drag** the Button down.
8. Release **Ctrl**.
9. Double-click the value of the Button's **Text** property.
10. Enter **Read Text**.
11. Press **Enter**.
12. Double-click the **Button**.
13. Click in the code.
14. Press **Ctrl+A** on your keyboard.
15. Press **Delete**.
16. Enter **message =**
17. Click **[]**.
18. Expand **Root Container**.
19. Expand **Text Field**.
20. Click **Text**.
21. Click **OK**.
22. Press **Enter**.
23. Type **system.gui.messageBox(message, title)**
24. Delete **, title**
25. Ensure your code looks like this:

```
message = event.source.parent.getComponent('Text Field').text
```

```
system.gui.messageBox(message)
```

26. Click **Apply**.
27. Move the **Component Scripting** dialog out of the way.
28. Enter something in the **Text Field**.

29. Click .
30. Click the **Button**.



You will see the text you entered in the message box.

Note: Rather than creating a variable for the message, then passing that text to the MessageBox function, it is possible to write that code on one line:

```
system.gui.messageBox(event.source.parent.getComponent('Text Field').text)
```

Component Extension Functions

Most of the more complex components have a set of Extension Functions, which are pre-defined functions you can use to expand on the functionality of the component.

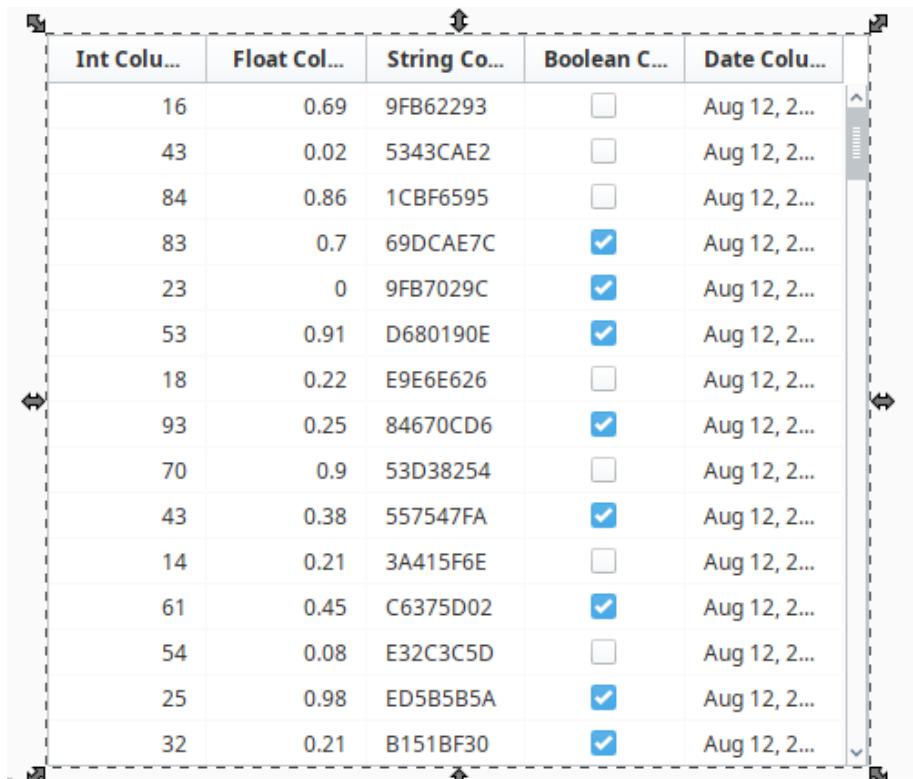
One useful extension function is in the Power Table, where we can fairly easily add code to change the table so that it has alternating row colors, which will make large tables easier to read.

1. Click **OK** to close the existing Component Scripting dialog.
2. Click .
3. Expand the **Components Palette**.
4. Click in the **Filter** Box.

5. Click .
6. Type **power**.
7. Drag a **Power Table** to the window.

There are a variety of ways to populate the Power Table with data, but for our purposes, we can going to use a feature of the table used for sales demos of Ignition.

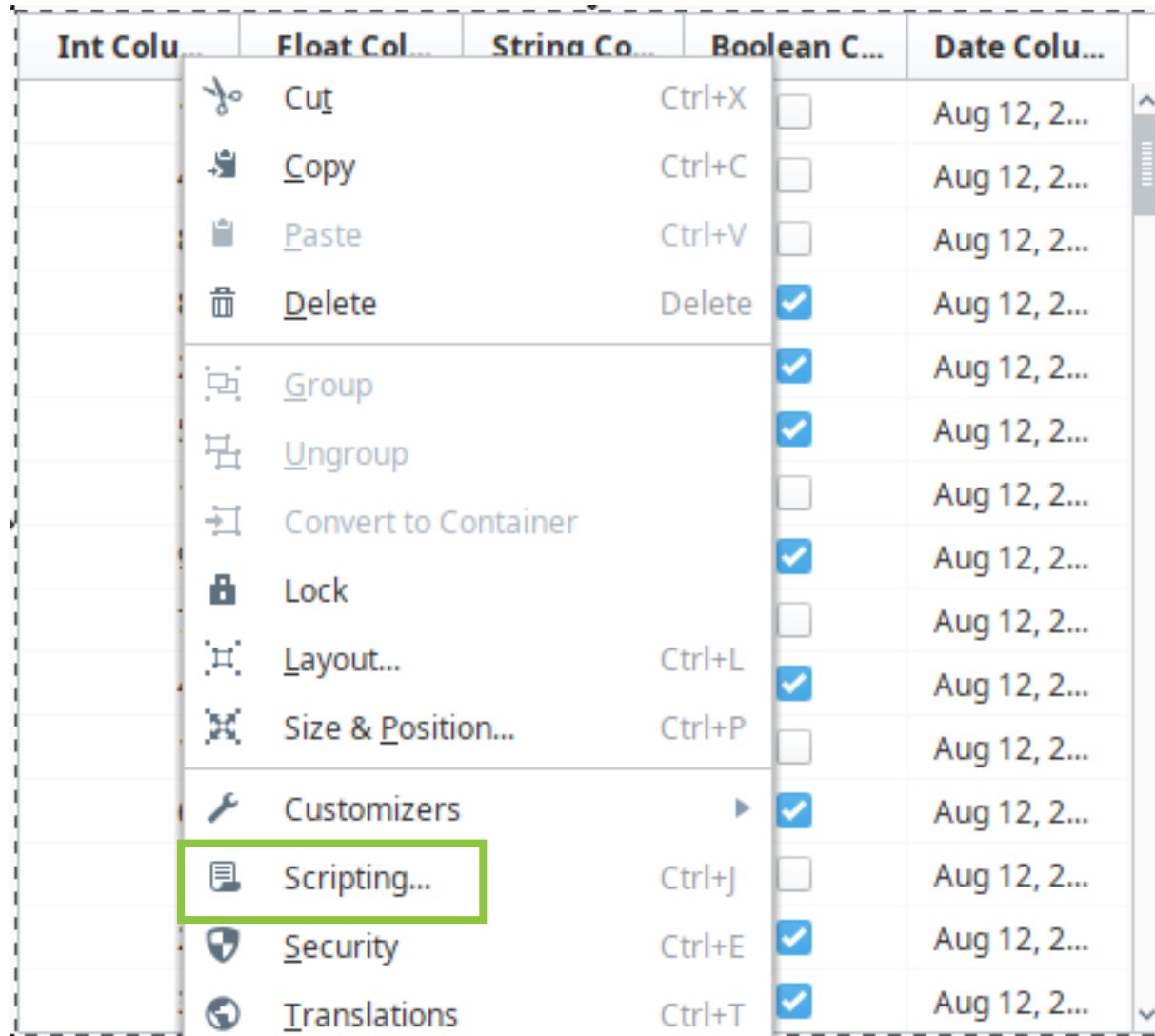
8. Scroll to the bottom of the property list.
9. Click **Test Data**.



Int Colu...	Float Col...	String Co...	Boolean C...	Date Colu...
16	0.69	9FB62293	<input type="checkbox"/>	Aug 12, 2...
43	0.02	5343CAE2	<input type="checkbox"/>	Aug 12, 2...
84	0.86	1CBF6595	<input type="checkbox"/>	Aug 12, 2...
83	0.7	69DCAE7C	<input checked="" type="checkbox"/>	Aug 12, 2...
23	0	9FB7029C	<input checked="" type="checkbox"/>	Aug 12, 2...
53	0.91	D680190E	<input checked="" type="checkbox"/>	Aug 12, 2...
18	0.22	E9E6E626	<input type="checkbox"/>	Aug 12, 2...
93	0.25	84670CD6	<input checked="" type="checkbox"/>	Aug 12, 2...
70	0.9	53D38254	<input type="checkbox"/>	Aug 12, 2...
43	0.38	557547FA	<input checked="" type="checkbox"/>	Aug 12, 2...
14	0.21	3A415F6E	<input type="checkbox"/>	Aug 12, 2...
61	0.45	C6375D02	<input checked="" type="checkbox"/>	Aug 12, 2...
54	0.08	E32C3C5D	<input type="checkbox"/>	Aug 12, 2...
25	0.98	ED5B5B5A	<input checked="" type="checkbox"/>	Aug 12, 2...
32	0.21	B151BF30	<input checked="" type="checkbox"/>	Aug 12, 2...

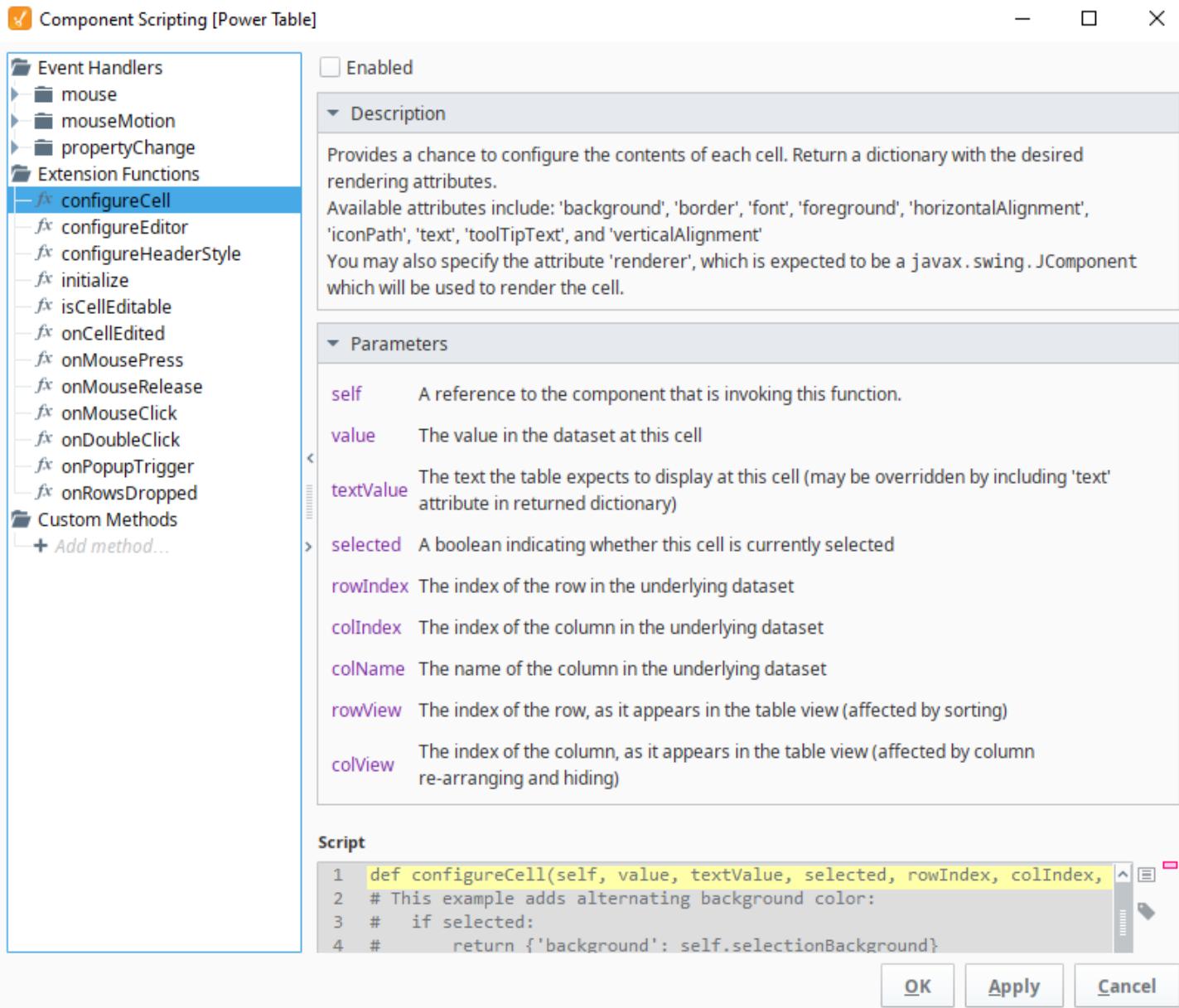
The **Test Data** property simply populates the table with random data, separated into columns based on data types.

10. Right-click the table.
11. Click **Scripting**.



The Component Scripting dialog now has a section of Component Functions.

1. Click **configureCell**.



Each Extension Function has a checkbox to enable it, a Description to explain what the function does, a list of the Parameters that are available to the function, and a Script section to type your code.

2. Click **Enabled**.
3. Collapse the **Description** section.
4. Click and drag to select **lines 3-8** of the sample code.
5. Press **Ctrl+/** on your keyboard to uncomment those lines.
6. Click **Apply**.
7. Move the **Component Scripting** dialog out of the way.

Int Colu...	Float Col...	String Co...	Boolean C...	Date Colu...
16	0.69	9FB62293	<input type="checkbox"/>	Aug 12, 2...
43	0.02	5343CAE2	<input type="checkbox"/>	Aug 12, 2...
84	0.86	1CBF6595	<input type="checkbox"/>	Aug 12, 2...
83	0.7	69DCAE7C	<input checked="" type="checkbox"/>	Aug 12, 2...
23	0	9FB7029C	<input checked="" type="checkbox"/>	Aug 12, 2...
53	0.91	D680190E	<input checked="" type="checkbox"/>	Aug 12, 2...
18	0.22	E9E6E626	<input type="checkbox"/>	Aug 12, 2...
93	0.25	84670CD6	<input checked="" type="checkbox"/>	Aug 12, 2...
70	0.9	53D38254	<input type="checkbox"/>	Aug 12, 2...
43	0.38	557547FA	<input checked="" type="checkbox"/>	Aug 12, 2...
14	0.21	3A415F6E	<input type="checkbox"/>	Aug 12, 2...
61	0.45	C6375D02	<input checked="" type="checkbox"/>	Aug 12, 2...
54	0.08	E32C3C5D	<input type="checkbox"/>	Aug 12, 2...
25	0.98	ED5B5B5A	<input checked="" type="checkbox"/>	Aug 12, 2...
32	0.21	B151BF30	<input checked="" type="checkbox"/>	Aug 12, 2...

Let's examine the code line-by-line to understand how it works.

8. Return to the **Component Scripting** dialog.

9. Click **configureCell**, if necessary.

Line 3 is an if statement, testing to see if the cell is selected.

Line 4 executes if the cell is selected, and returns a background of the selectedBackground, which is a property of the table that defaults to light blue.

Line 5 is an else if statement that looks at the rowView of the cell. If you look up at the Parameters, you will see that the rowView parameter is the index of the current row in the table. Because it is based on what is displayed on the table, and not on the underlying dataset, it is effected by sorting, which means that if a user clicks a header cell to resort the table, the rowView index numbers will remain the same. In other words, the first visual row of the table is always rowView 0, and second is 1, and so forth.

This line then uses the modulo operator to see what the value of the remainder is if we divide by 2. This will be either 0 or 1, depending on if the row is odd or even. In this case, we're seeing if it's 0, and returning a background of white on row 6. See “Basic Math” on page 359 for a review of the modulo operator.

Line 7 is the else: if the row is not selected, and its rowView is not even, this line executes, and results in line 8 returning a background of gray.

Note: #DDDDDD is a hexadecimal code for a color. See “Editing Colors” on page 113. A feature of hexadecimal is that anytime all six digits are the same, you get a shade of gray.

The sample code provided by the extension function lets us set up alternating rows without typing any code, but let's expand on this a bit further. In addition to the alternating rows, we also want to highlight certain rows based on the value in the Int Column. Specifically, we want to make any row have a red background if the value of the Int Column on that row is over 50.

To figure out if the row matches our criteria, we need to get the value of the cell in the Int Column. To do this, we're going to use the self parameter of the function. As you can see from the Parameters list, self is a reference to the component that is calling this function—in our case, the Power Table. From there, we can access the table's data property, and use the getValueAt() function to return a specific value. Within that function, we can reference rowIndex, which, according to the Parameters, is a reference to a row in the underlying dataset of the table, and then provide a reference to the column by its name. That combination of a row and a column gives us a specific cell in the table. We can then compare that value to the given criteria.

1. Click at the **end of line 4**.
2. Press **Enter**.
3. Press **Backspace**.
4. Enter **elif self.data.getValueAt(rowIndex, "Int Column") > 50:**
5. Press **Enter**.
6. Press **Tab**.
7. Enter **return {'background': 'red'}**
8. Ensure your code looks like this. Pay close attention to indentation.

```
# This example adds alternating background color:
```

```
    if selected:
```

```

        return {'background': self.selectionBackground}

    elif self.data.getValueAt(rowIndex, "Int Column") > 50:

        return {'background': 'red'}
```

elif rowView % 2 == 0:

```

        return {'background': 'white'}
```

else:

```

        return {'background': '#DDDDDD'}
```

9. Click **OK**.

Int Colu...	Float Col...	String Co...	Boolean C...	Date Colu...
16	0.69	9FB62293	<input type="checkbox"/>	Aug 12, 2...
43	0.02	5343CAE2	<input type="checkbox"/>	Aug 12, 2...
84	0.86	1CBF6595	<input type="checkbox"/>	Aug 12, 2...
83	0.7	69DCAE7C	<input checked="" type="checkbox"/>	Aug 12, 2...
23	0	9FB7029C	<input checked="" type="checkbox"/>	Aug 12, 2...
53	0.91	D680190E	<input checked="" type="checkbox"/>	Aug 12, 2...
18	0.22	E9E6E626	<input type="checkbox"/>	Aug 12, 2...
93	0.25	84670CD6	<input checked="" type="checkbox"/>	Aug 12, 2...
70	0.9	53D38254	<input type="checkbox"/>	Aug 12, 2...
43	0.38	557547FA	<input checked="" type="checkbox"/>	Aug 12, 2...
14	0.21	3A415F6E	<input type="checkbox"/>	Aug 12, 2...
61	0.45	C6375D02	<input checked="" type="checkbox"/>	Aug 12, 2...
54	0.08	E32C3C5D	<input type="checkbox"/>	Aug 12, 2...
25	0.98	ED5B5B5A	<input checked="" type="checkbox"/>	Aug 12, 2...
32	0.21	B151BF30	<input checked="" type="checkbox"/>	Aug 12, 2...

The table will now display any row where the Int Column value is greater than 50 in red. Other rows will continue to alternate between white and gray.

Client and Gateway Event Scripts

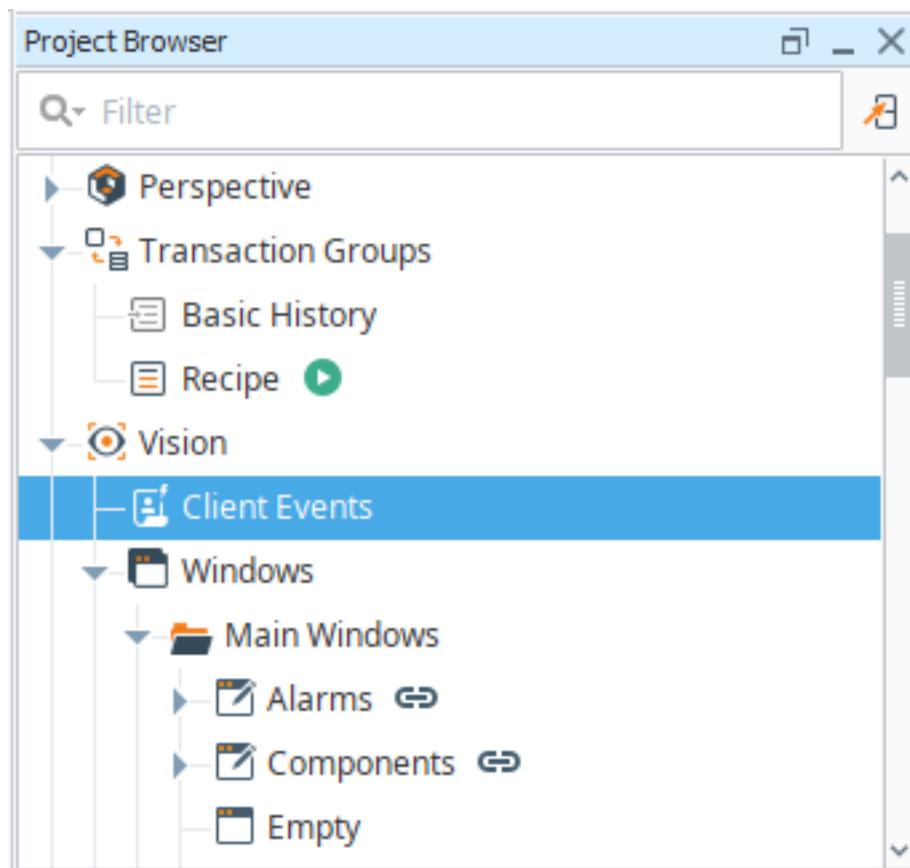
Client and Gateway events are special events not based around components. Many times, these scripts can execute automatically, without the user needing to perform some action to trigger them.

Client Events

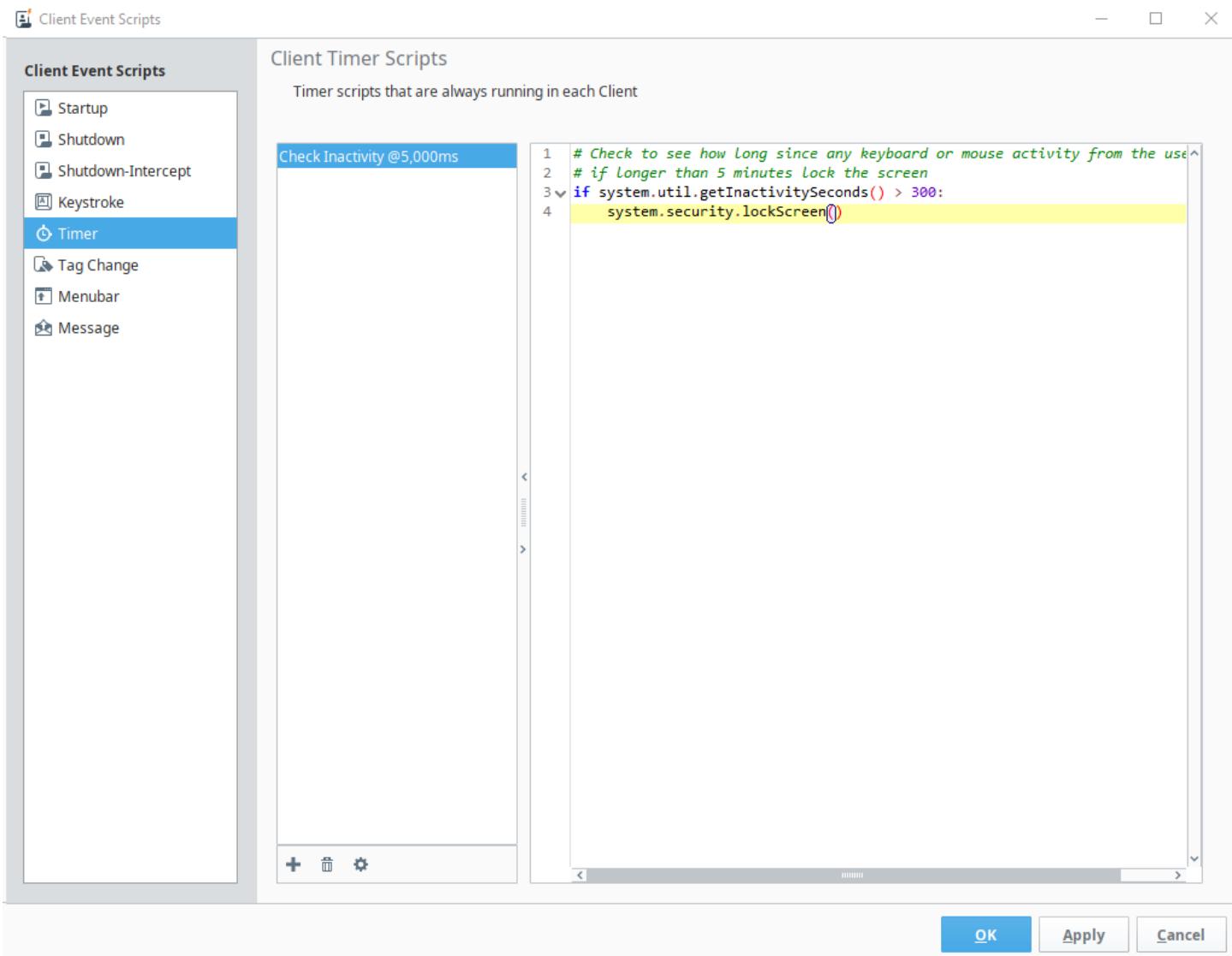
There are a handful of Client Events that are triggered in the Vision client in response to actions. Like scripts on components, each script can fire on each client that is currently running this project. A script executing in the Client is considered to be in the Client Scope, which means that each client is running a separate instance of the script.

Let's open the Client Events Scripts dialog box to explore an example.

1. Double-click **Client Events** in the Project Browser.



2. Click **Timer**.
3. Click **Check Inactivity @5,000ms**



This script is part of the project template that we selected when we first created our project. It runs every five seconds, and checks how long the client has been inactive. If it has been inactive for more than five minutes, then it locks the client. This is why we have to log into the client every time we leave it alone. We could change the number of seconds—the 5000 at the end of line 3—if we wanted a different inactive time. We can disable the script altogether, and thus making it so that we do not need to log into to client at all (except when we are opening it) by double clicking on the script name and unchecking the Enabled checkbox.

Gateway Events

Gateway Events work a little differently than Client Events, even though many of them seem similar. While the Client Events are triggered by things happening in the Client, the Gateway Events are triggered by things running in the Gateway. This means that these

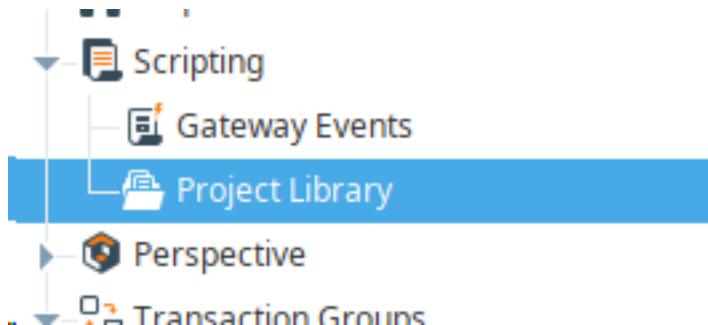
scripts execute in the Gateway Scope. By running in the Gateway Scope, these scripts can't directly affect objects within the Client such as windows and components, but they also continue to run even if no clients are open.

Gateway Events are accessed from the Scripting module in the Project Browser.

Project Script Library

The Project Script Library is a place where you can create your own script functions. These functions can be called from anywhere in the project, and are helpful when you have scripts that you need to use repeatedly.

1. Expand **Scripting** in the Project Browser.
2. Click **Project Library**.



3. Enter **myScripts** in the Name of the script field.
4. Click **Create**.

Create a New Script

5. Enter **def myMessage(message):**
6. Press **Enter**.

7. Enter `system.gui.messageBox(message)`

8. Ensure your code looks like this:

```
def myMessage(message):  
    system.gui.messageBox(message)
```

9. Click **File**.

10. Click **Save All**.

11. Return to the **Scripting** Window in Vision.

12. Click a **Button**.

13. Press and hold **Ctrl** on your keyboard.

14. Drag the **Button** to create another copy.

15. Release **Ctrl**.

16. Double-click the current value of the Button's **Text** property.

17. Enter **Project Script**.

18. Press **Enter**.

19. Double-click the **Button**.

20. Click in the code.

21. Press **Ctrl+A** on your keyboard.

22. Press **Delete**.

23. Enter `myScripts.myMessage("This is my Project Script")`

24. Click **OK**.

25. Click .

26. Click the **Button**.

In this example, we simply made a function that calls the built-in system function to open a dialog box. That's not terribly useful, but it hopefully gives you the idea that you can put your code in the Project Scripting library and then use it anywhere else in your project.

Perspective

Perspective is a module that allows you to create resources that enable a user to easily transition from Viewing the project on a computer to Viewing it on a mobile device. Everything is based around web technologies such as HTML, CSS, and JavaScript, which allow you to build screens that are displayed in the web browser and can easily respond to changing screen sizes. Perspective is a different visualization system than Vision, utilizing different layouts that determine where components are on the screen, and how they resize. Because Perspective is launched in the web browser, you essentially build web pages that the user can navigate to with a URL. These pages can then have many Views on them, and each View is made up of a series of components.

Perspective and Web Terms

There are some important terms to understand in working with Perspective. As we will be using these terms throughout the remainder of this chapter, we want to define them here.

View The basic thing we create in Perspective are Views. They are similar conceptually to both Windows and Templates in Vision: we define our interface as a set of Views, which in turn are made up of components and other, embedded Views.

Page A Page in Perspective is a View that has an assigned Page URL and can therefore be viewed directly from a web browser.

Session When we launch Perspective in the browser, we are launching a Session. Conceptually, Sessions are similar to the Client in Vision, and can be thought of in terms of how the end user interacts with our Perspective application.

HTML HTML stands for Hypertext Markup Language, and is used to define the structure of a web page. Every web page you have ever visited has been composed in HTML. While Perspective does ultimately translate components to HTML, you cannot edit HTML directly in Perspective.

CSS Cascading Style Sheets is the standard for formatting on the web. All formatting, positioning, sizing, and layout of components and Views in Perspective is done with CSS. A considerable amount of what you do in Perspective requires working, both directly and indirectly, with CSS.

JavaScript A scripting language, conceptually similar to Python, that is widely used on the web. Like HTML, Perspective relies heavily on JavaScript, but does not require that you directly interact with it. Scripting in Perspective, while not covered in this course, is still done in Python.

JSON JSON, which stands for JavaScript Object Notation, is a lightweight, plain-text format for describing data. Almost all data in Perspective, including component properties, is handled with JSON.

Creating a View

A View allows us to visualize components on the screen and display data. At a basic level, a View in Perspective is similar to a Window in Vision.

However, the process of creating a View is different. Whereas in Vision we simply need to decide if we are going to create a Main Window, a Popup Window, a Docked Window,

or a Template, in Perspective we always simply create Views. But there are three very important decisions we need to make when we first create the View.

Name

The name of the View is the same as the Name of a Window in Vision. You can give your Views any name that makes sense.

Root Container Enter

Because Perspective relies on web technologies, layout is dramatically more complex than it is in Vision. When creating a View, you need to decide which container type you will use, which will dictate how you lay out the components in the View and how the View will behave in the browser window.

Note: You cannot change the Root Container Enter after you create a View. Always carefully consider the needs of the View before selecting a container type.

While there are a total of six container types available in Perspective, we are only going to explore three of them in this course:

Coordinate Container This container allows you to place components in arbitrary locations in the View and freely move and resize them. This container most closely mimics the way layout is handled in Vision.

Breakpoint Container The Breakpoint Container allows you to embed Views at specific screen widths, and have the browser automatically switch between them. See "Breakpoint Layout" on page 429.

Flex Container This container relies on an underlying web technology known as Flexbox. It allows you to create Views that can work on the widest variety of screen sizes, but is also the most complex. See ""Understanding the Flex Container" on page 404.

Page URL

To access a View in the browser, it must have a URL. Remember that in Perspective, we are essentially creating a web site.

Every site needs a home page, which is a page that can be accessed by just typing the URL of the web site. For example, you can access the home page of our web site by visiting inductiveautomation.com.

In Perspective, the URL of the home page will be set to a slash. Any other page needs to have a URL unique to the site.

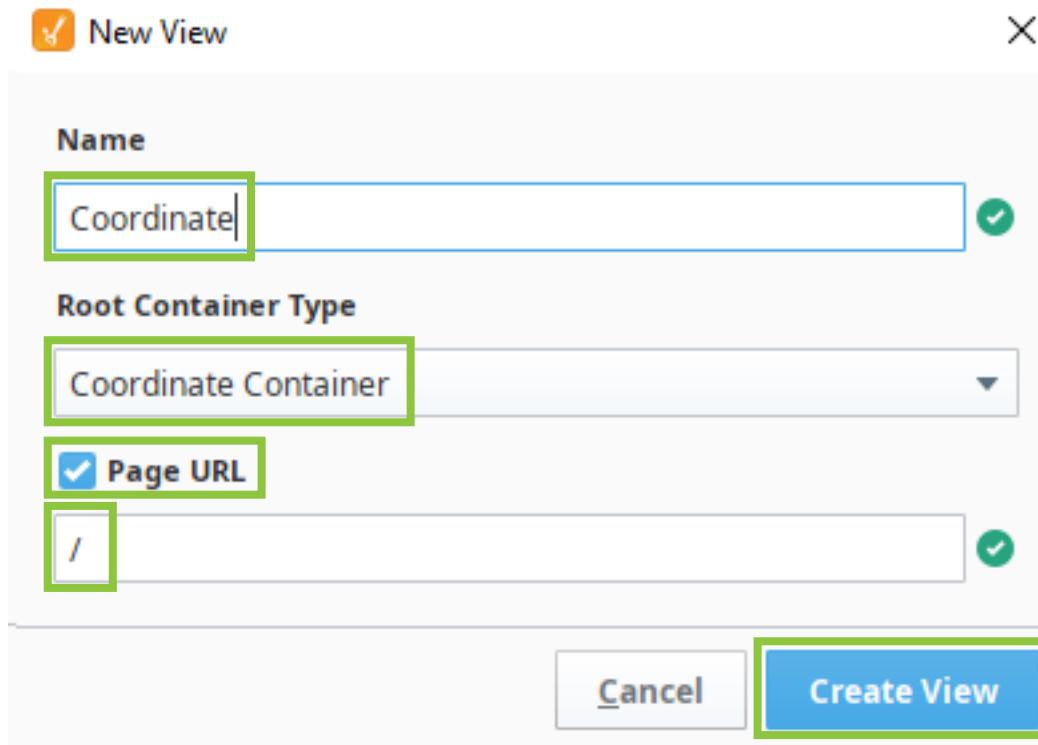
It is also possible to create a View that you intend to only embed within other Views, and not be directly accessible in a browser. In that case, can you opt not to set a page URL.

Unlike the Root Container Enter, you can change your mind later: you can add or remove URLs on pages at any time.

Creating Our First View

Let's begin exploring how Perspective works by creating a View that will be very similar to the first Window we created in Vision.

1. Click **Perspective** in the Project Browser.
2. Click **Create New View**.
3. Enter **Coordinate** in the Name field.
4. Ensure Root Container Type is set to **Coordinate Container**.
5. Check **Page URL**.
6. Ensure the field contains only a **/**.
7. Click **Create View**.



Now that we have the View created, we can start adding components.

You will see that the layout of the Perspective module is like what we customized the Vision layout to look like: we have our Project Browser and Tag Browser on the left, and the Property Editor and Components Panel on the right. By default, the Components Panel collapses.

8. Mouse over **Perspective Components** to expand the panel.
9. Click in the **Search** box.
10. Enter **tank**.
11. Drag a **Cylindrical Tank** to the View.
12. Click the Search box.
13. Click .
14. Enter **slider**.
15. Drag a **Slider** to the View.
16. Click the **Search** box.
17. Click .
18. Enter **label**.
19. Drag a **Label** to the View.
20. Click **File**.
21. Click **Save All**.

Positioning and Sizing Components

When using the Coordinate Container, you can freely drag components to any spot on the View. In this respect, the Coordinate Container works similarly to Vision Windows if layout is set to Anchored by default.

There are, however, a few key differences.

Smart Guides

Unlike in Vision, Perspective includes a feature known as Smart Guides. As you drag a component in the View, you will see red guides appear that help you to line up components with each other and with the View as a whole. These will greatly speed up your design, as aligning and spacing components is much easier.

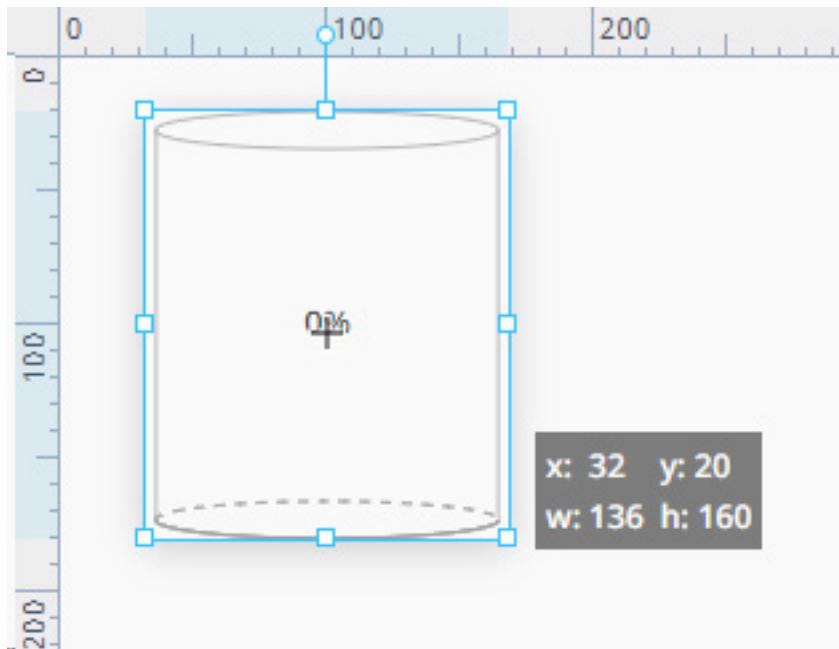
Anchors and Rotation

Every component that you add to a View in a Coordinate Container has a cross-hair in its center. You may accidentally drag this instead of the component when trying to move things, particularly on smaller components like Labels.

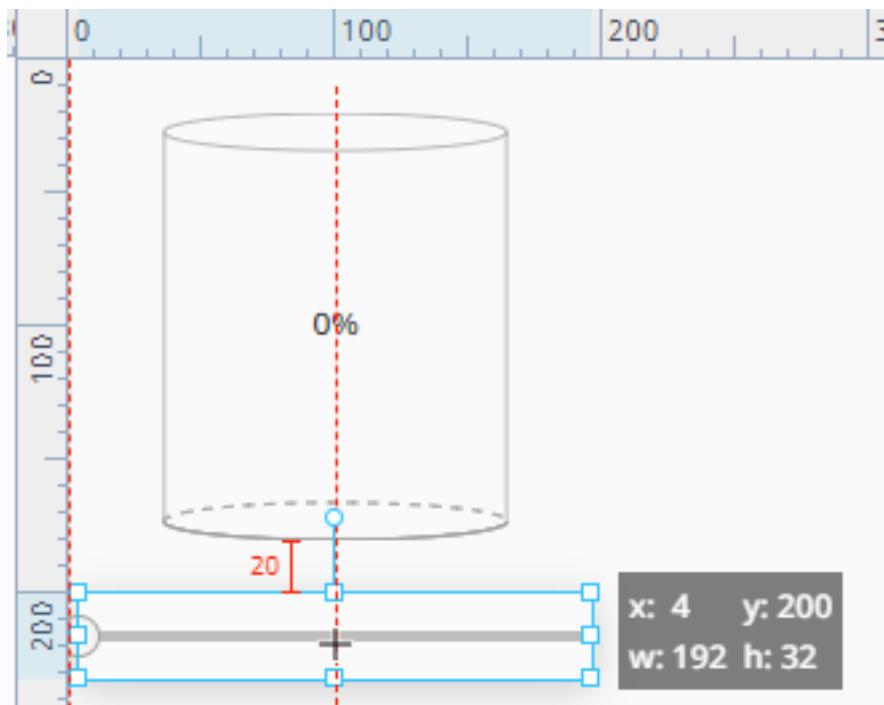
This cross-hair is the component's Anchor. It is the point around which an component will rotate. All components can be freely rotated by clicking and dragging the handle that rises out of the top of the component. While it may appear at first that objects rotate around their center, in fact they are rotating around their anchor. Drag the anchor to another spot—it doesn't even need to be on the component itself—and then rotate the component again.

Let's set up our three components so that they are nicely aligned and spaced.

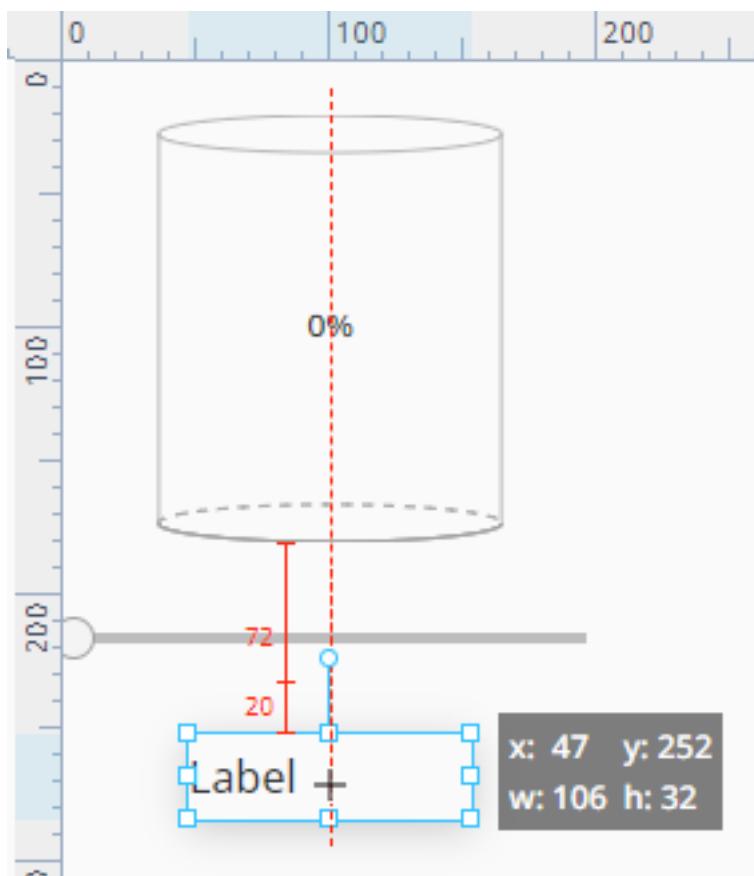
1. Drag the **Cylindrical Tank** to that its center is 100 pixels from the left edge of the View and its top is 20 pixels from the top edge of the View.



2. Drag the **Slider** so that it is **centered** and **20 pixels** below the Tank.



3. Drag the right edge of the **Label** to the right to make it approximately twice as wide as it started.
4. Drag the **Label** so that it is **centered** and **20 pixels** below the Slider.



Coordinate Layout Scaling

The coordinate container by default in a **Fixed** mode, which works similarly to Vision's anchored mode. Each component will have a set x and y location and widths and heights that do not change, regardless of the size of the View.

In **Percentage** mode, the layout works similarly to Vision's relative mode. The components will be given x, y, width, and height that is a percentage of its position on the View, which allows it to grow and shrink with the View.

You can change this mode by clicking on the background of the View, or on the View in the Project Browser, then change the property on the Property Editor.

Bindings in Perspective

Fundamentally, creating bindings between components in Perspective is the same as it is in Vision. There are just a few minor interface differences.

Perspective Properties

The Perspective Property Editor is more logically organized than the Property Editor in Vision. Every component has its properties divided into several categories, just like in Vision, but all components have the same set of categories, and within those categories, properties are organized based on how frequently we think they'll get used, so the properties you need will generally be towards the top of the list.

Props The basic properties of the components. The vast majority of the time, the property you are looking for will be there.

Position The properties needed to position the component. In a Coordinate Container, you will see properties such as width, height, x and y, and rotation. Other container types will have other properties in this category.

Custom Just like in Vision, you can add custom properties to components.

Meta Other properties, such as the Name of the component, can be found here. We will not explore these properties in this course, although they can be useful in more advanced projects.

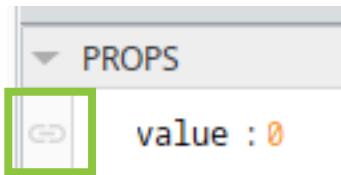
Adding Bindings

In Vision, every property has  button to the right of its value. At first glance, this button appears to be missing in Perspective, but it is there. You will see an apparently

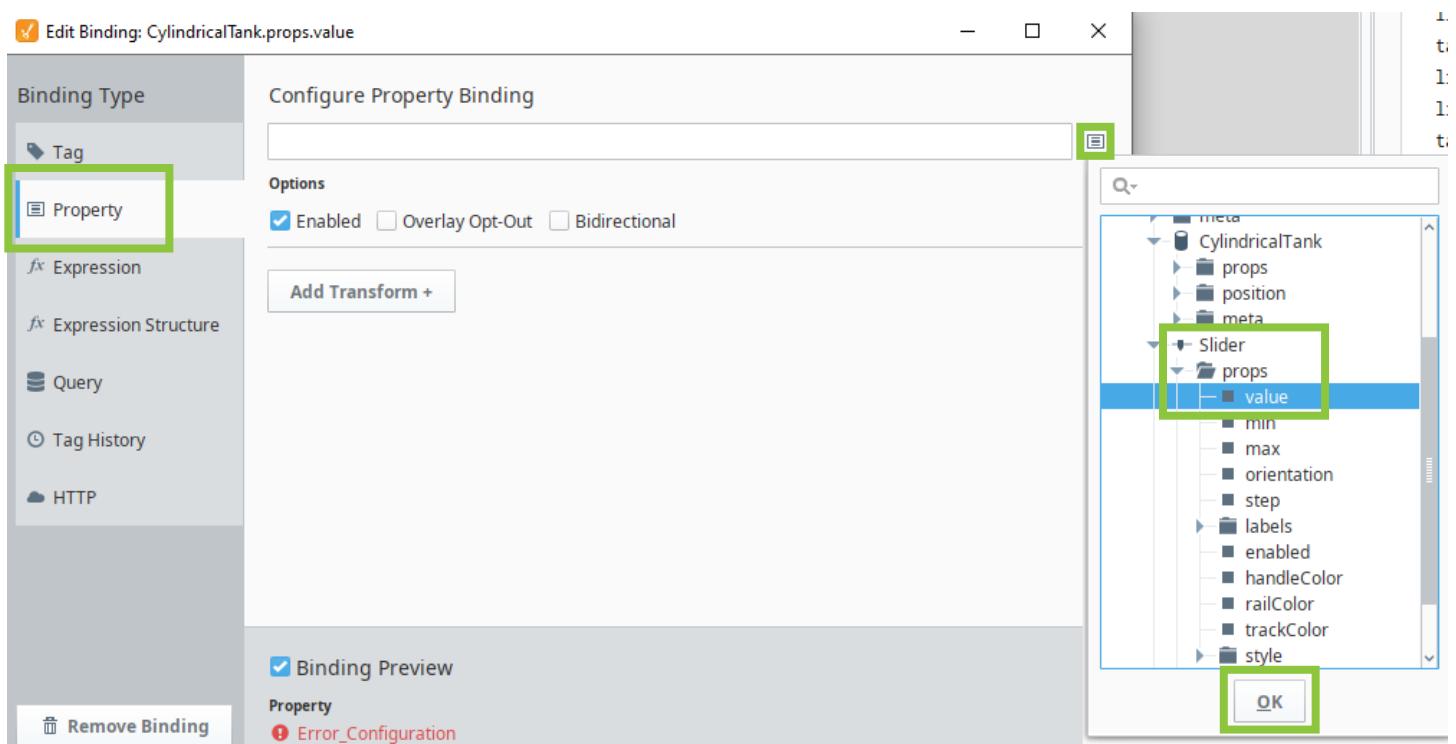
empty column running down the left side of the Property Editor. If you move your mouse over this column for any property, the  button will appear. Once a binding has been added to a property, the icon will appear at all times.

Let's go ahead and bind the Tank's value to the Slider's value, and also get the Label to display the value with an expression, just like we did in Vision.

1. Click the **Tank**.
2. Click  for the Tank's **value**.



3. Click **Property**.
4. Click .
5. Expand **root**.
6. Expand **slider**.
7. Click **value**.
8. Click **OK**.



9. Click **OK**.

Note: The Edit Binding dialog box in Perspective has a great additional feature: Binding Preview. Always check this to make sure that the binding is working before clicking OK.

10. Click the **Label**.
11. Click  for the Label's text property.
12. Click **Expression**.
13. Click .
14. Expand **root**.
15. Expand **slider**.
16. Click **value**.
17. Enter `+ " gallons"`
18. Ensure your expression looks like this:

```
{.../Slider.props.value} + " gallons"
```

19. Click **OK**.
20. Click .
21. Drag the **slider**.

Both the tank and the label will change accordingly.

22. Click **File**.
23. Click **Save As**.

Tag Bindings

Tag bindings work the same way that they do in Vision, and we can utilize a lot of the drag and drop functionality that we are used to. Just as with Vision, you can drag and drop a Tag directly onto a View, or you can add a component and drag and drop a Tag on it, or you can add a component and manually create the binding. Unlike in Vision, though, Perspective does not add additional bindings when you drag and drop a Tag directly onto the View, so there is not appreciable difference between any of these methods.

Previewing in the Browser

Just as it is a good idea to test Vision Windows directly in the Client, you will want to test your Perspective pages in a browser.

You can do this from Designer, just like you can in Vision.

1. Click **Tools**.
2. Click **Launch Perspective...**
3. Click **Launch Session**.

Whichever page you have designated with a Page URL of / will open.

Notice the web address we're visiting:

<localhost:8088/data/perspective/client/CoreClass>

The first part of that is, of course, our Gateway address. After that, all Perspective sessions will have /data/perspective/client. That will be followed by the name of your project, and then the page URL. Modern browsers hide a slash at the end of a URL, but it is there.

Perspective Views as Templates

Technically, Perspective does not have Templates in the same sense that Vision does. Instead of creating something that is specifically a Template, in Perspective any View can be embedded in another View. Thus, we get the functionality of Templates without the limitations.

Often, you will not assign a Page URL to a View that you intend to use as a Template, although assigning a Page URL does not limit the functionality of the View in any way.

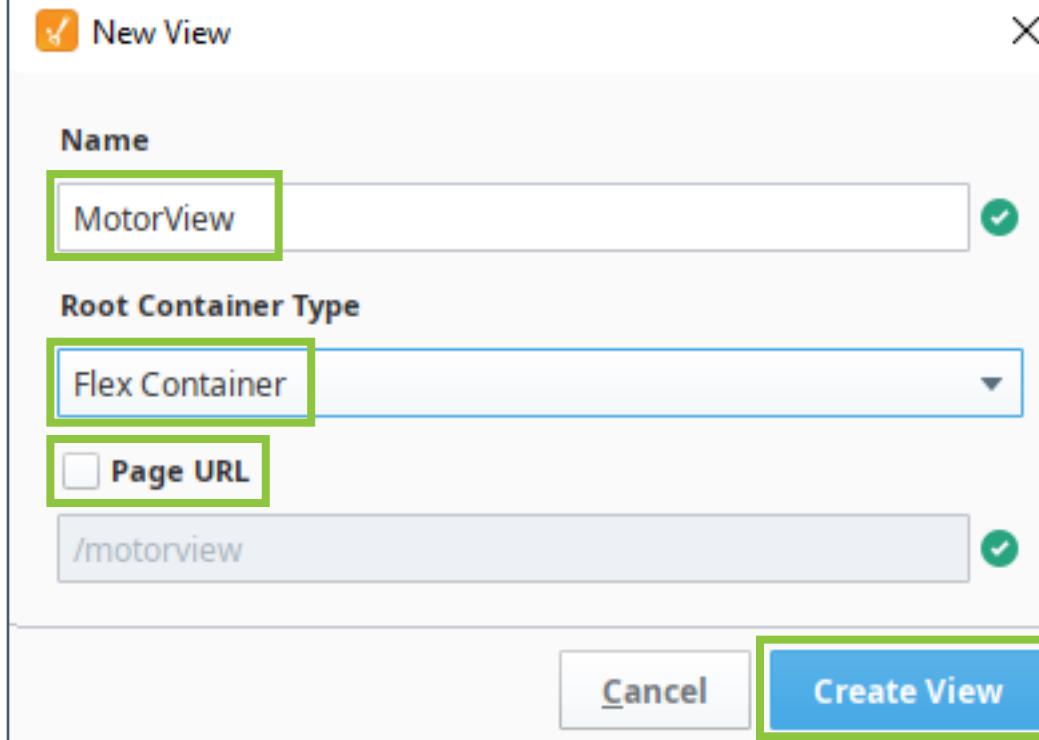
You can use any Root Container Enter for Views that you plan to use as Templates. In our example below, we will use a Flex Container.

We are going to create a View for our Motors that we can then embed in other Views.

1. Click **Perspective** in the Project Browser.
2. Click **Create New View**.
3. Enter **MotorView** in the Name field.
4. Select **Flex Container** in the Root Container Type list.

Note: Remember that you cannot change the Root Container Enter after the fact. Be certain you have the correct option selected here.

5. Leave Page URL **unchecked**.
6. Click **Create View**.



Understanding the Flex Container

The Flex Container is based on the CSS technology known as Flexbox. It was created in response to the need to create responsive layouts—web layouts that can work on a variety of different screen sizes. Hence its name: it creates flexible layouts that can rearrange and resize parts of a page to fit on larger or smaller displays.

Since its release in 2009, Flexbox has become a standard for page layout on the web.

Flex allows you to create one-dimensional layouts by laying out components in either rows or columns. As such, you cannot directly drag components to specific positions when using Flex; every component must be next to each other component, either horizontally or vertically. The direction is determined by the Direction property, which can be found at the top of the Property Editor.

Each Flex container has two axes: a main axis and a secondary axis. If the Direction is set to row, the main axis is horizontal and the secondary vertical; if set to column, the

axes are reversed. All components are laid out along the main axis, and aligned on the secondary axis.

You also cannot directly resize components when using Flex. Instead, all component sizing is handled via the following three properties, all of which are found in the Position category of the Property Editor.

Grow This property allows a component to grow along the main axis, both in initial sizing and if the container grows in response to being displayed in a larger window. The value of Grow is a unitless, positive integer. If set to 0, the component will not grow. If set to any other number, the container will attempt to distribute the size of the container as a ratio of each component's Grow value. In other words, if there are three components in a container, and each has a Grow of 1, they should all equally sized. If one has a Grow of 1 and the other two are each 2, then the first will be given $\frac{1}{3}$ of the space of the container, and the other two half of the remaining space.

Shrink Shrink works the same as Grow, but in the opposite direction: if the container gets smaller, the space for the components will be redistributed based on this value.

Basis The starting size of the component, before remaining space is distributed. This can be set as a specific length, using any valid CSS unit of measurement (most often, pixels or percents.) If set to 0, extra space around the component will not be factored into its size; if set to auto, it will be.

Note: All components default to a Grow of 0, Shrink of 1, and Basis to an arbitrary size determined by the developers.

Add Components

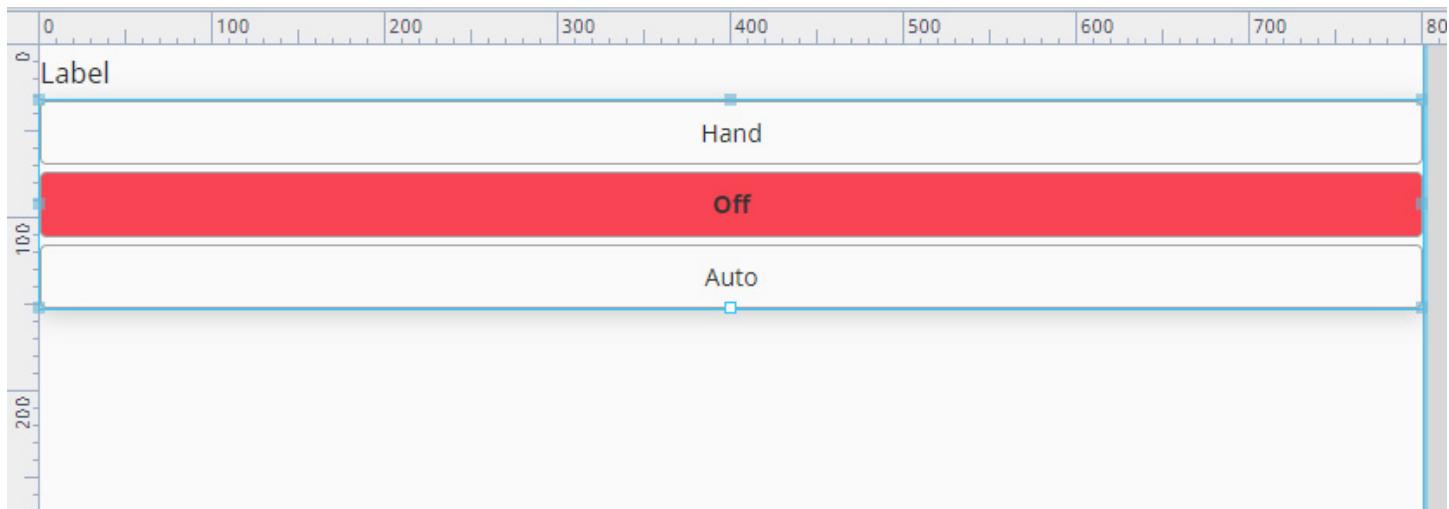
Now that we have our View with the proper root container, let's add components to it.

1. Click **Perspective Components**.
2. Enter **label** in the Search box.
3. Drag a **Label** into the View.

Because this is a Flex Container, it does not matter where you drop the Label—it will automatically snap to the top of the container and stretch across its width.

4. Click **Perspective Components**.
5. Click  to clear the previous search.
6. Enter **multi** in the Search box.

7. Drag a **Multi-State Button** into the View.



8. Click **File**.
9. Click **Save All**.

Nested Containers

As stated above, the Flex Container allows for one-dimensional layout: everything must either be in a single row or a single column. But of course, there are plenty of times when you want more complex layouts. In our case, having the first two components stack vertically is fine, but below the Multi-State Button, we want a Label next to an LED. To do this, we need to nest a container.

All the Perspective Root Container types also exist as components that can be nested within our layouts. Any container type can be nested within any other container type, but some care should be given to consider how they might interact. For example, the Flex container is designed to allow its components to grow and shrink as the size of the outer container changes. The Coordinate container, on the other hand, when set to Fixed mode, is designed to be a static layout that will not change as the screen size changes. Therefore, nesting a Flex container inside a Coordinate container allows you to use the flexible layout properties of Flex, but you lose the flexible scaling.

1. Click **Perspective Components**.
2. Enter **flex** in the Search box.
3. Drag **Row** to the View.

Note: You will notice that there appear to be three Flex components: Flex, Row, and Column. These are all the same. Row and Column are just copies of the Flex component with the Direction property preset to the stated value. Many components in Perspective show multiple copies like this, but it never matters if you drag the preset to the View, or drag a different variant and then change the property.

Deep Selection

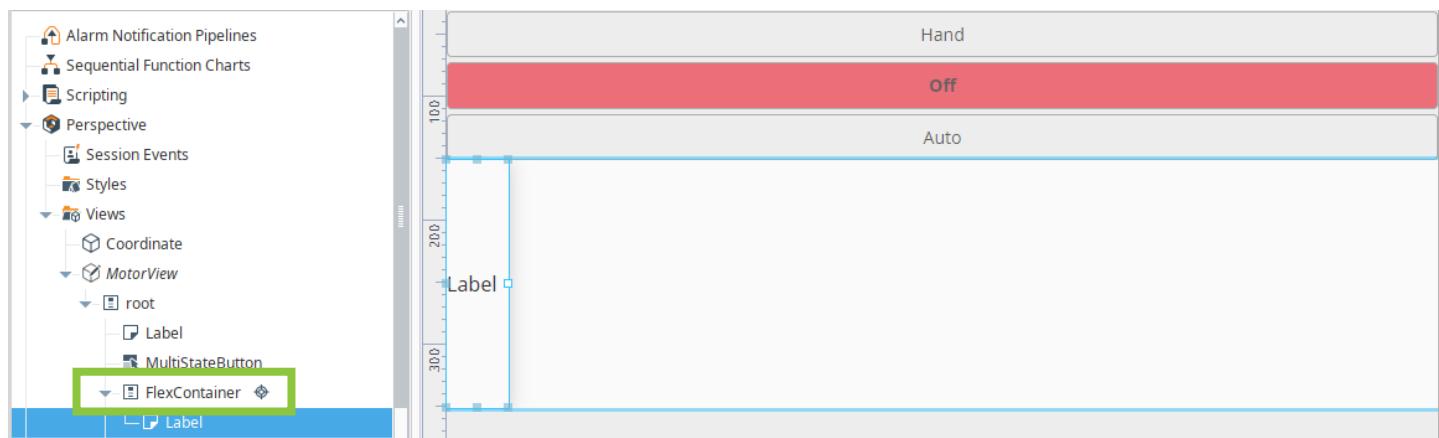
Once you begin nesting containers, you will run into a problem: we need to explicitly tell Perspective which container we want to set properties on and add new components to.

The Project Browser displays a crosshair icon——next to a container as a visual indicator as to which container is currently deeply selected. The container's properties are reflected at the top of the Perspective Property Editor, and new components will be added to that container.

You can deeply select a new container by right-clicking it and clicking Deep Select. You can also double-click the container in the View or click it while press the Shift and Alt keys on your keyboard.

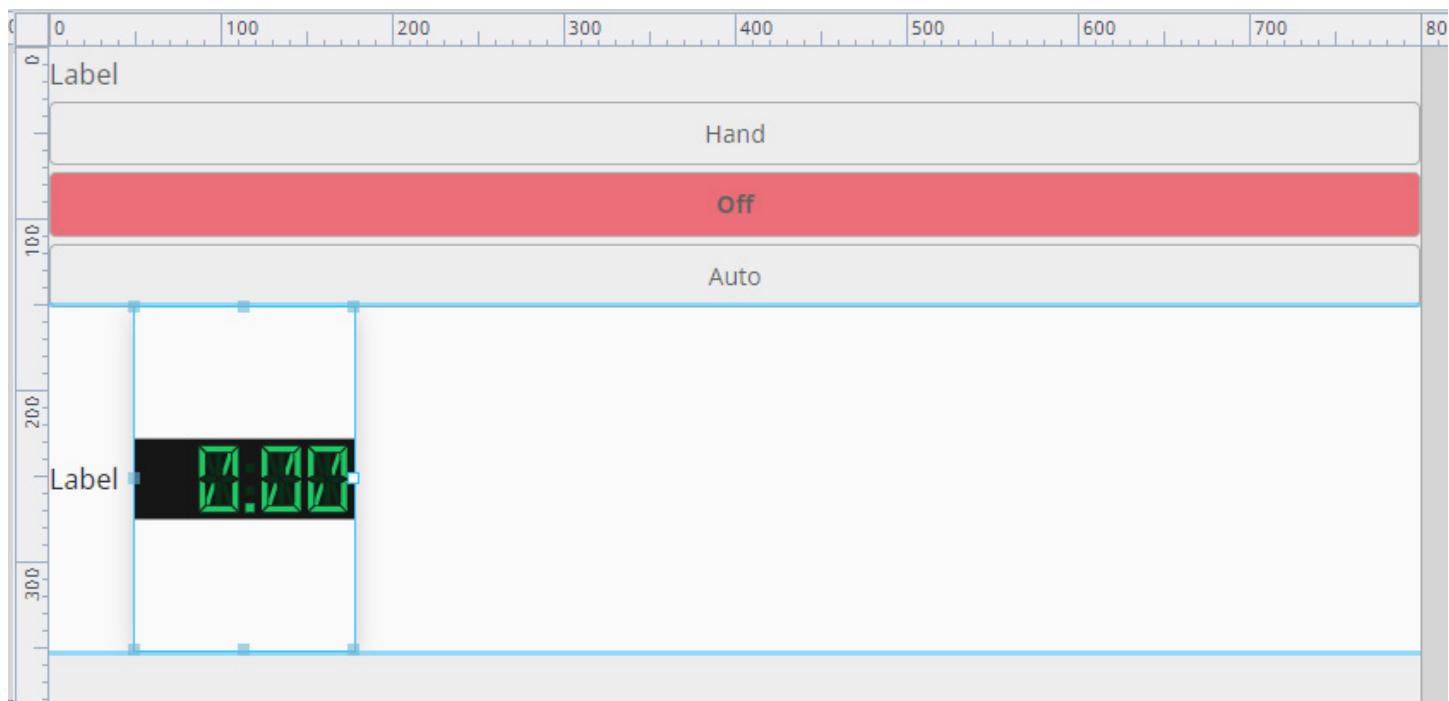
Once a container is deeply selected, the outer container and its contents will grey out, providing an additional visual clue that you have the correct container selected.

1. Double-click the new **Flex Container** to deep select it.
2. Click **Perspective Components**.
3. Enter **label** in the Search box.
4. Drag a **Label** to the nested container.

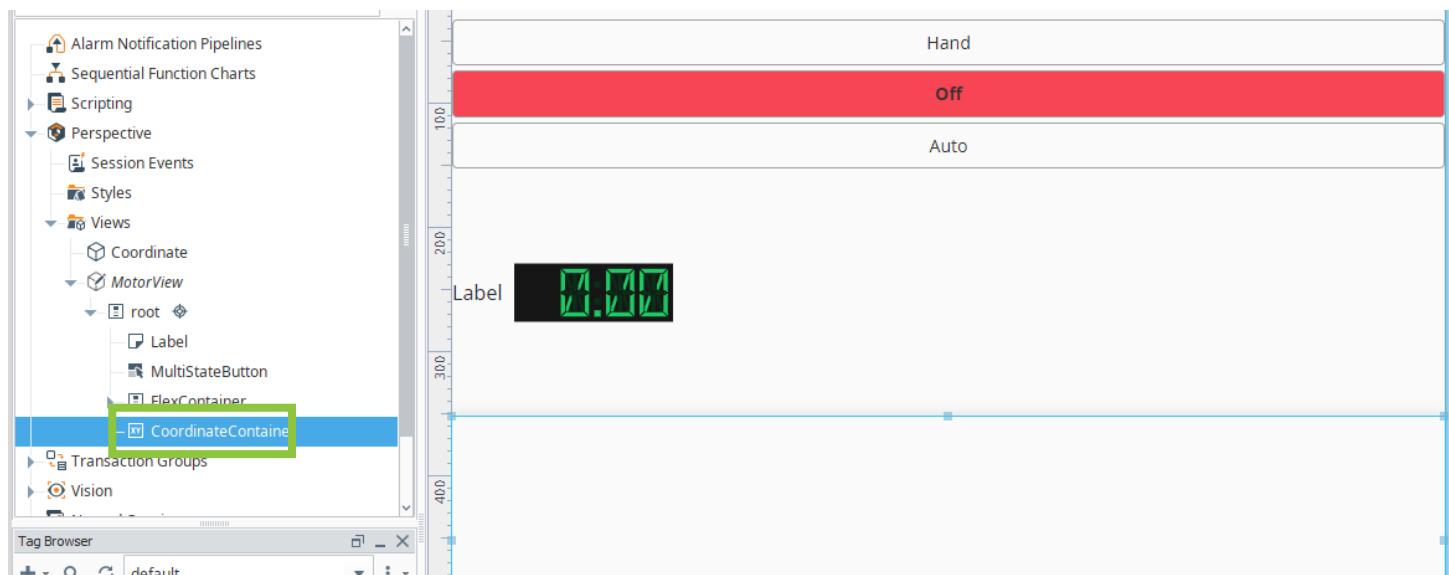


5. Click **Perspective Components**.

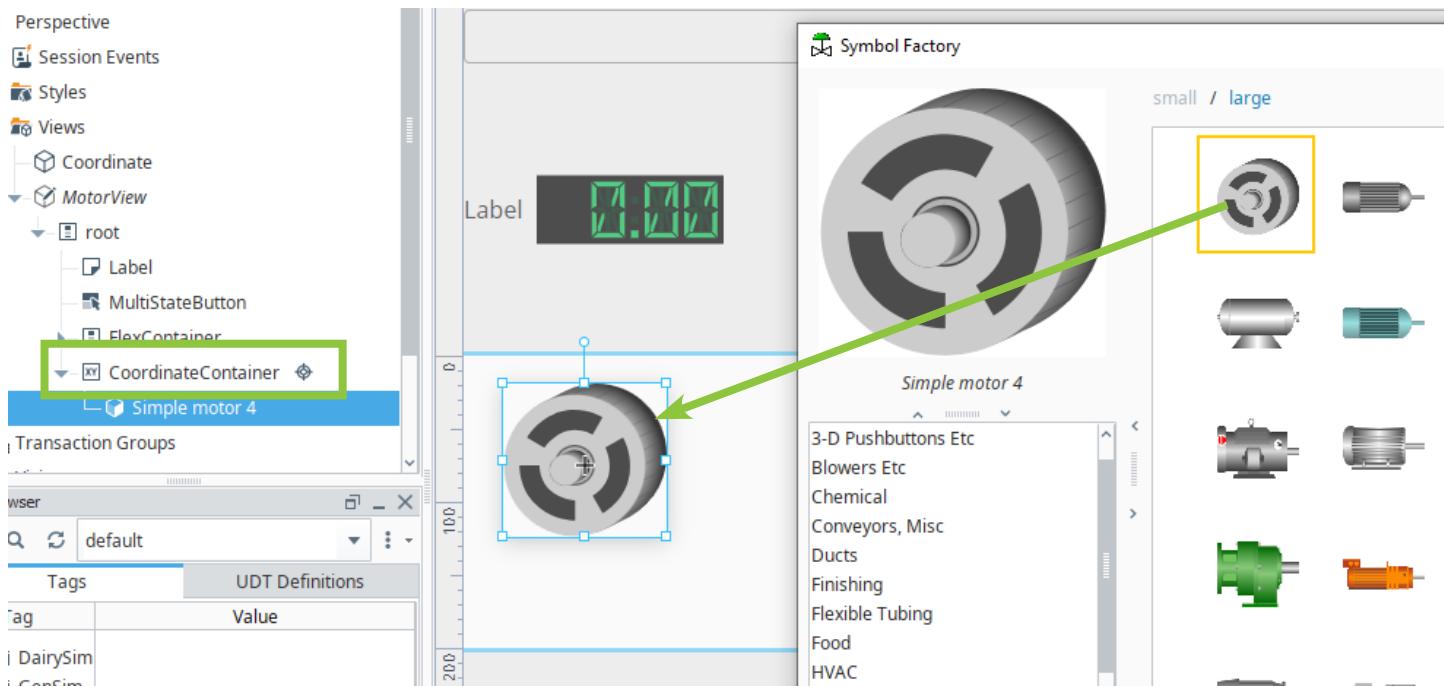
6. Enter **led** in the Search Box.
7. Drag an **LED Display** to the nested container.



8. Double-click outside of the nested container. Ensure that **root** has  on the Project Browser.
9. Click **Perspective Components**.
10. Enter **coordinate** in the search box.
11. Drag a **Coordinate** container to the View.



12. Double-click the **Coordinate** container to Deep Select it.
13. Click **Tools**.
14. Click **Symbol Factory**.
15. Click **Motors**.
16. Drag **Simple Motor 4** into the Coordinate container.



17. Close the Symbol Factory.
18. Click **File**.
19. Click **Save All**.

Perspective View Parameters

When we created this View, the intent was to embed it in other Views; in essence, treat it as a Template. In Vision, we were able to assign parameters to be passed into Templates to control what they display. In Perspective, we can do the same.

Note: Perspective parameters are set on the View itself, not its root container.

1. Click **MotorView** on the Project Browser.
2. Click **Add View Parameter** on the Perspective Property Editor.

Custom Property Types

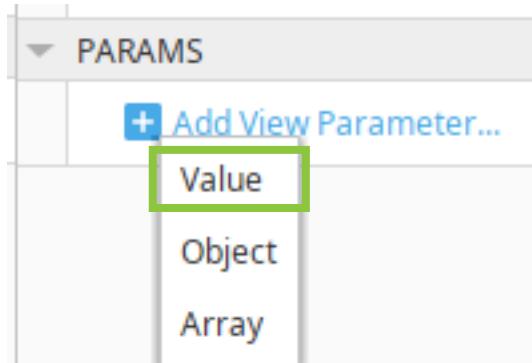
At first glance, adding parameters to Perspective seems more complex than Vision. Rather than simply typing the name of the parameter and setting its datatype, we have three seemingly odd choices when adding a parameter in Perspective. These choices are because the underlying data for all properties in Perspective is not Java as it is in Vision, but rather, JSON.

Value A simple value, such as we have been using for parameters throughout the course. You will need to provide a key—the name of the parameter—and a default value.

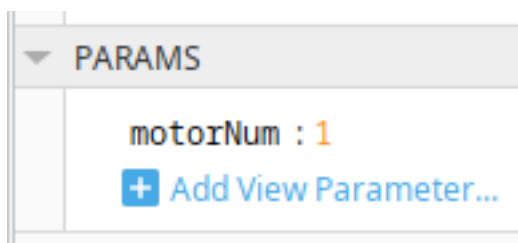
Object A complex data type that can contain multiple keys and values. Using objects as parameters is beyond the scope of this course.

Array Another complex type, where values are referenced by position rather than name. Conceptually similar to lists in Python, but like Objects, using Arrays as parameters are beyond the scope of this class.

1. Click **Value**.



2. Enter **motorNum** in the key field.
3. Press **Tab**.
4. Enter **1** in the value field.

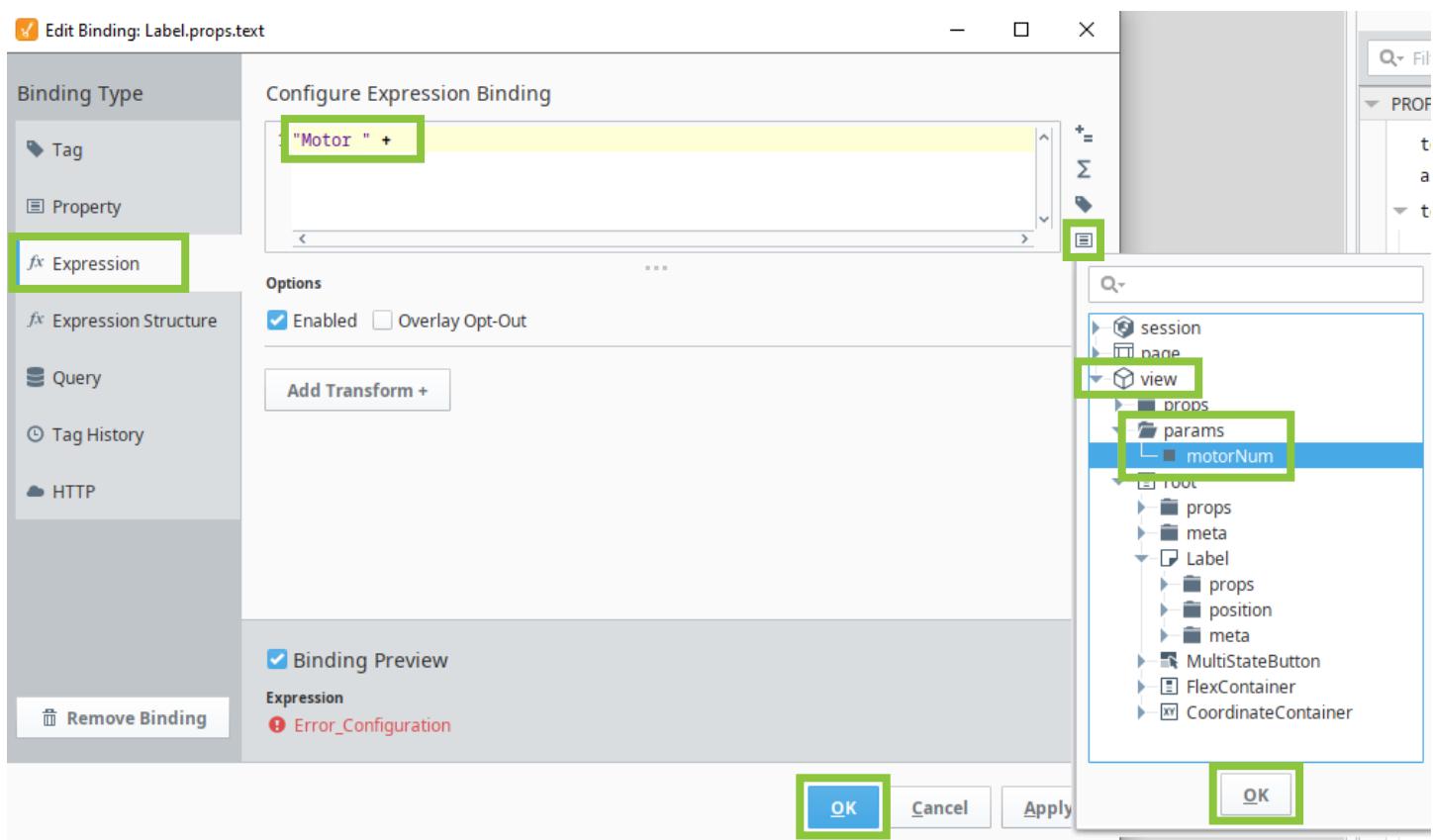


Perspective Expressions

Expressions in Perspective are essentially the same as they are in Vision. There is a slight difference in how the path to the View Parameter looks, but as we will be selecting that from a dialog box rather than typing it, that difference does not really matter too much.

1. Click the **Label** at the top of the View.
2. Click  for the Label's **text** property.
3. Click **Expression**.
4. Enter **"Motor " +**
5. Click .
6. Expand **View**.
7. Expand **params**.
8. Click **motorNum**.
9. Click **OK**.
10. Ensure your expression looks like this:

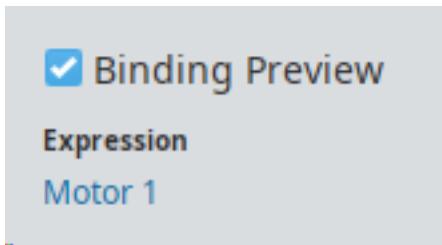
```
"Motor " + {View.params.motorNum}
```



Previewing Bindings

One very helpful feature of the Perspective Edit Binding dialog box is the Binding Preview at the bottom, which lets you confirm that your binding is correct before you close the dialog box. Be sure to get in the habit of checking this when working in Perspective, as it will save you a lot of time.

1. Confirm that the **Binding Preview** shows **Motor 1**.
2. Click **OK**.

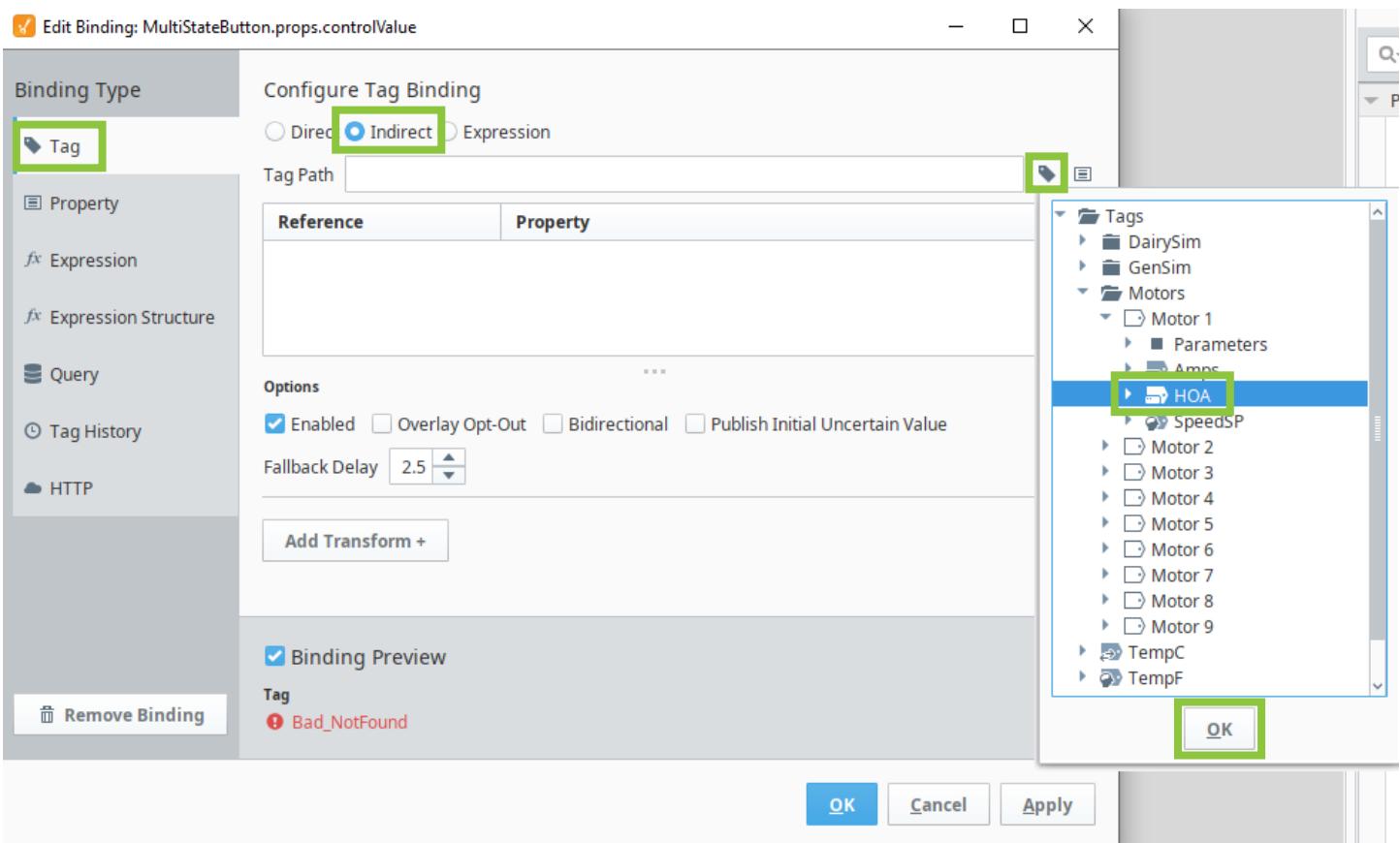


Perspective Indirect Tag Bindings

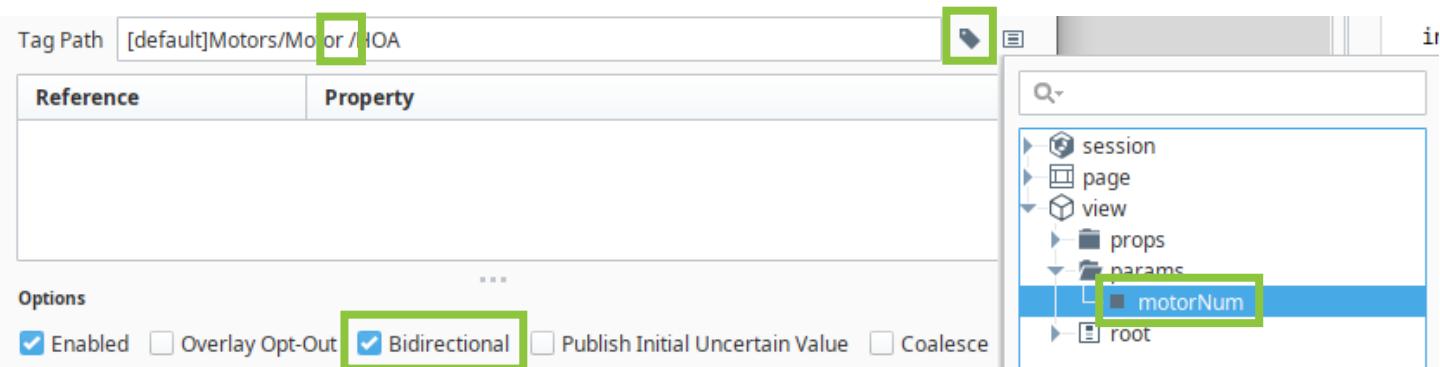
Just as in Vision Template, Perspective Views can use indirect Tag bindings to bind based on parameters. See "Indirect Tag Bindings" on page 167 for a review of indirect bindings.

There is a slight interface difference between Perspective and Vision when creating indirect bindings, and the way that the path to the parameter is specified is different as well, but the concept behind what we are trying to achieve is the same.

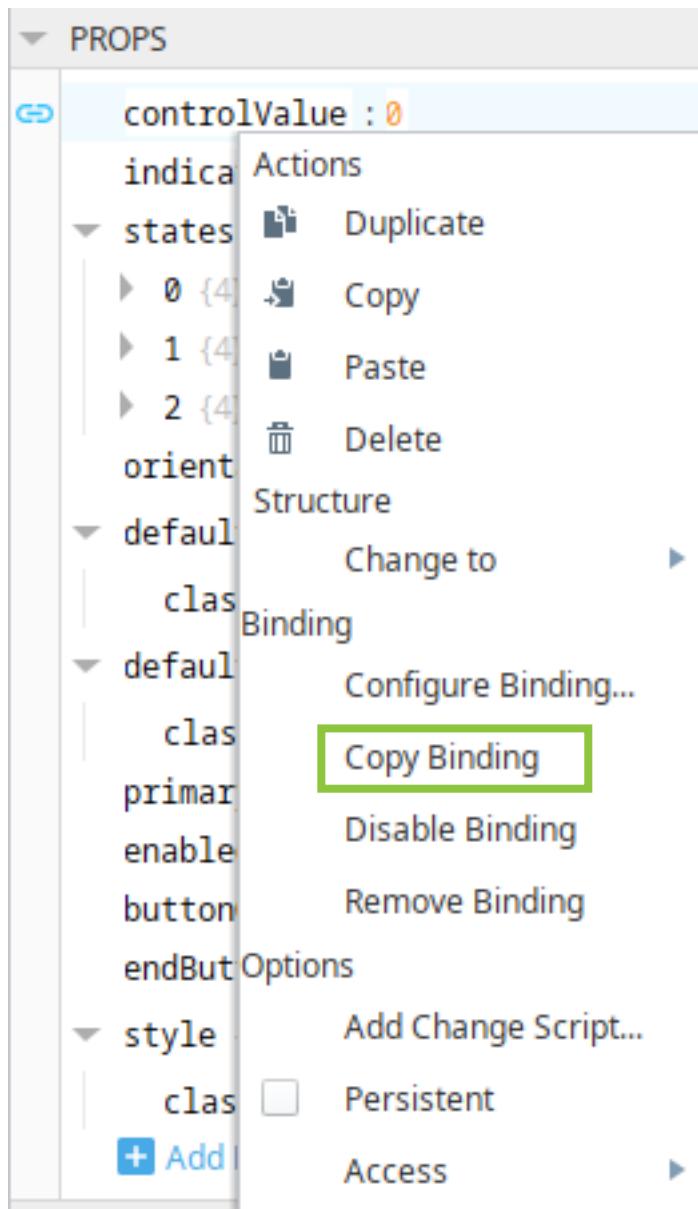
1. Click the **Multi-State Button**.
2. Click for controlValue.
3. Click **Tag**.
4. Click **Indirect**.
5. Click .
6. Expand **Tags**.
7. Expand **Motors**.
8. Expand **Motor 1**.
9. Click **HOA**.
10. Click **OK**.



11. Click in the Tag path.
12. Delete **1**.
13. Click .
14. Expand **View**.
15. Expand **params**.
16. Click **motorNum**.
17. Click **OK**.
18. Check **Bidirectional**.



19. Confirm the **Binding PreView** shows a value of 0, 1, or 2.
20. Click **OK**.
21. Right-click **controlValue**.
22. Click **Copy Binding**.



Note: Be sure to click Copy Binding, not Copy. Because properties in Perspective are stored in JSON files, any property can be copied and pasted. In this case, we need to ensure that we only copy the binding, not the entire controlValue property.

23. Right-click **indicatorValue**.
24. Click **Paste Binding**.

25. Double-click the **nested Flex container** with the Label and LED.
26. Click the **Label**.
27. Click in the **text** property value to edit the text.
28. Enter **Amps**.
29. Click the **LED display**.
30. Click  for the value property.
31. Click **Tag**.
32. Click **Indirect**.
33. Click .
34. Expand **Tags**.
35. Expand **Motors**.
36. Expand **Motor 1**.
37. Click **Amps**.
38. Click **OK**.
39. Click in the **Tag path**.
40. Delete **1**.
41. Click .
42. Expand **View**.
43. Expand **params**.
44. Click **motorNum**.
45. Click **OK**.

Confirm the Binding PreView shows a proper value. It should be a large decimal number that is updating every second.

46. Click **OK**.

Adjusting Layout Properties

Now that we have all our components in place and the bindings set up, we want to adjust the properties of the Components and nested Containers so that the layout looks good.

Remember that the three properties we need to adjust are **Grow**, **Shrink**, and **Basis**.

Review the definitions above if you need to.

In our case, we are going to set both Grow and Shrink to the same value, so that their proportions continue to work properly whether we expand or shrink the window. We will also adjust Basis as needed.

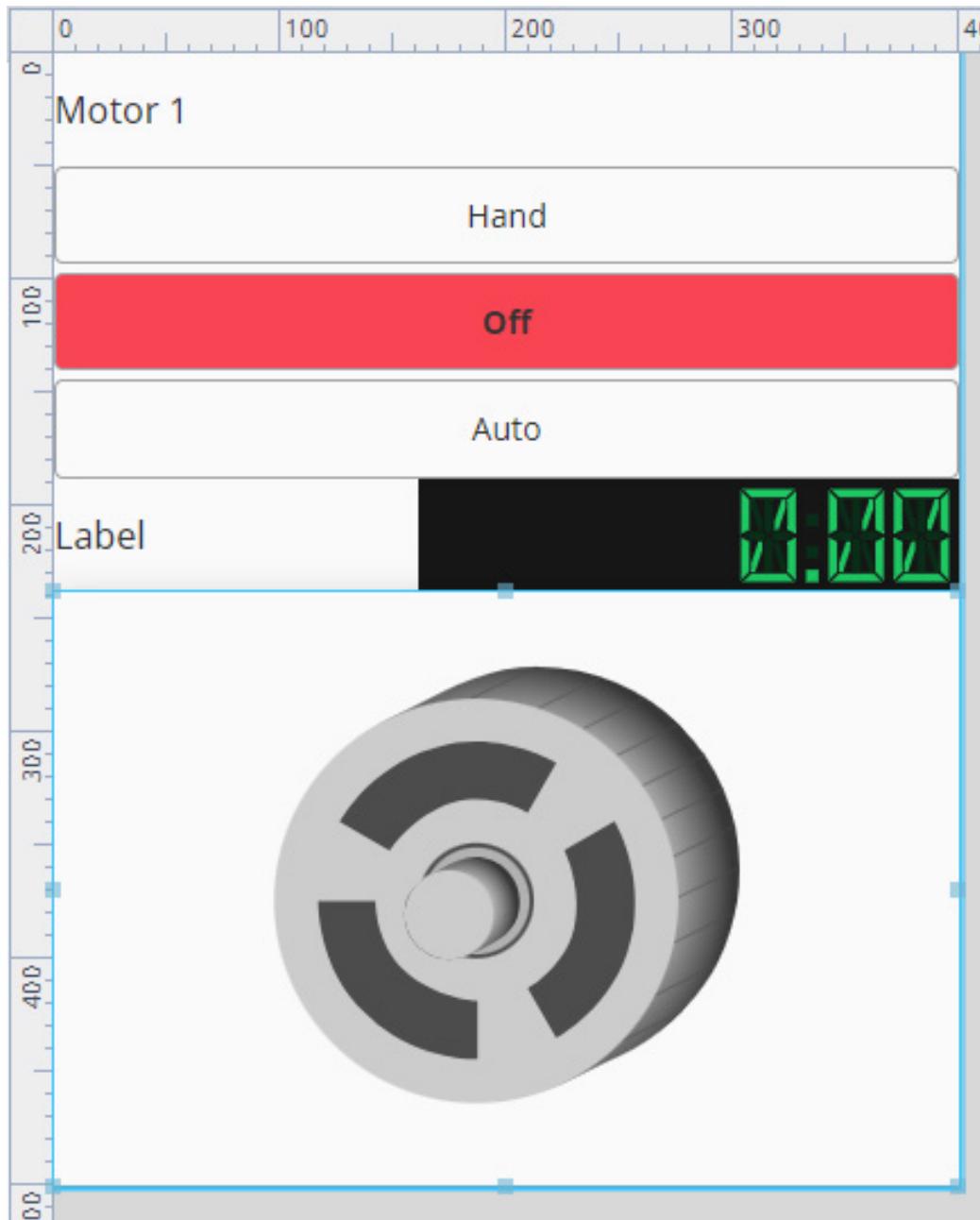
1. Click **MotorView** in the Project Browser.
2. Enter **400** in the **width** property.
3. Enter **500** in the **height** property.
4. Click the **Label** showing the motor name.
5. Enter **0** in the **shrink** property.
6. Enter **50px** in the **basis** property.



Note: Be sure to type px for the units. There are no default units in Perspective, so we need to ensure that we enter them when necessary.

7. Click the **Multi-State Button**.
8. Enter **1** in the **grow** property.
9. Enter **75px** in the **basis** property.
10. Click the nested **Flex Container**.
11. Enter **0** in the **shrink** property.
12. Enter **50px** in the **basis** property.
13. Double-click the nested **Flex Container**.
14. Click the **Label**.

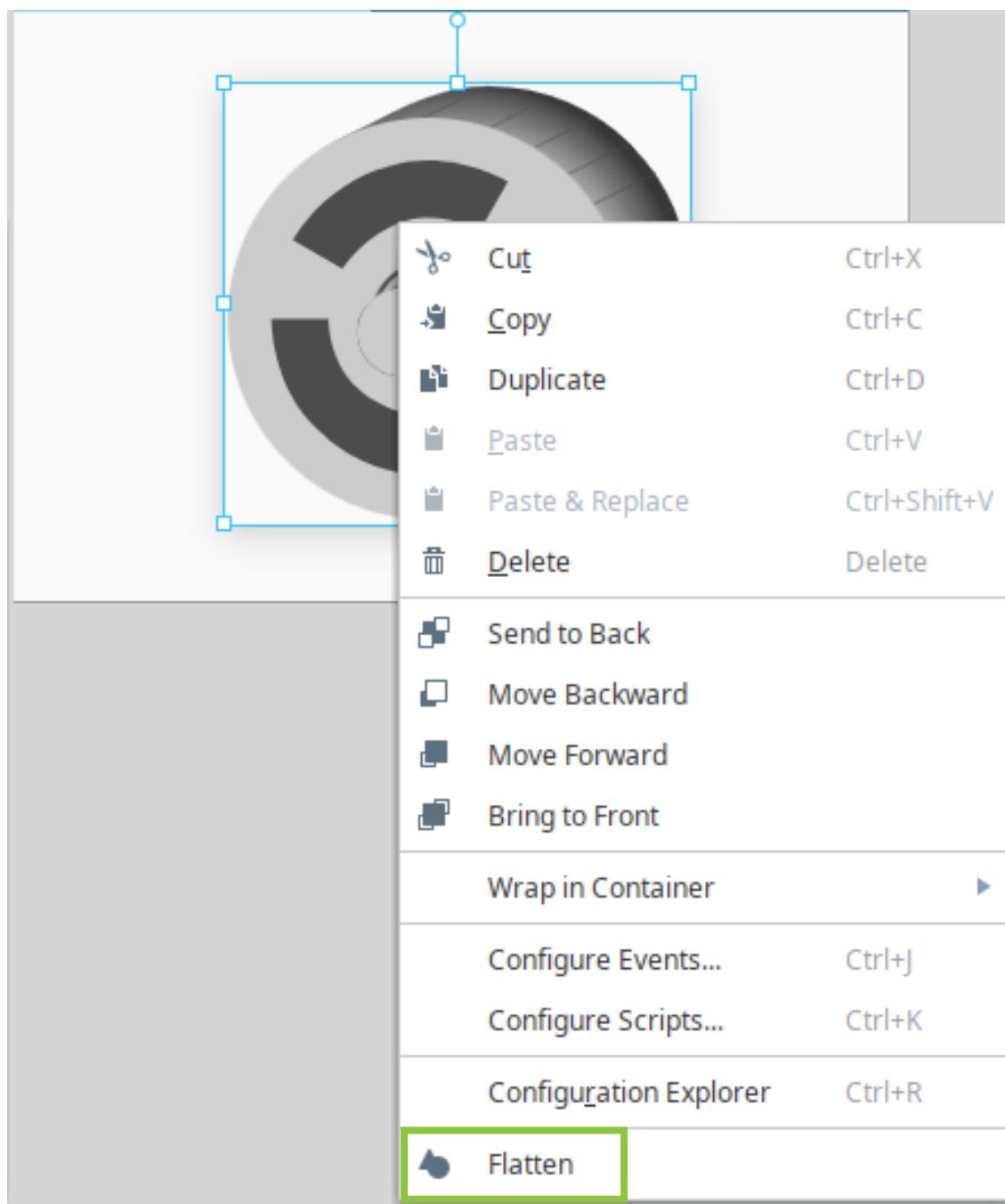
15. Enter 1 in the **grow** property.
16. Click the **LED Display**.
17. Enter **1** in the **grow** property.
18. Click outside the nested Flex Container.
19. Click the nested **Coordinate Container**.
20. Enter **1** in the **grow** property.
21. Double-click the nested **Coordinate Container**.
22. If necessary, **drag and resize the motor** to visually center it within the Coordinate Container.



Color Overlay

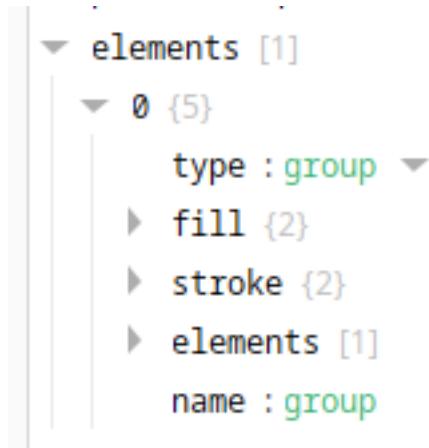
The final step in setting up this Template is to add the color overlay on the motor image, just as we did in Vision. When we added the motor, we placed it in a Coordinator container. While we could have added the Symbol Factory image directly to the View, doing so would not have allowed us to place a copy of the image on top of the original, a necessary step in setting up the color overlay. Only a Coordinate container allows us to place components on top of other components like this.

1. Right-click the **motor image**.
2. Click **Flatten**.



This creates a copy of the image that is solid, but semi-transparent, grey. You'll see in the Project Browser that there are now two Simple motor components.

3. Click **Simple motor 5** in the Project Browser.
4. Expand **elements** in the **Perspective Property Editor**.



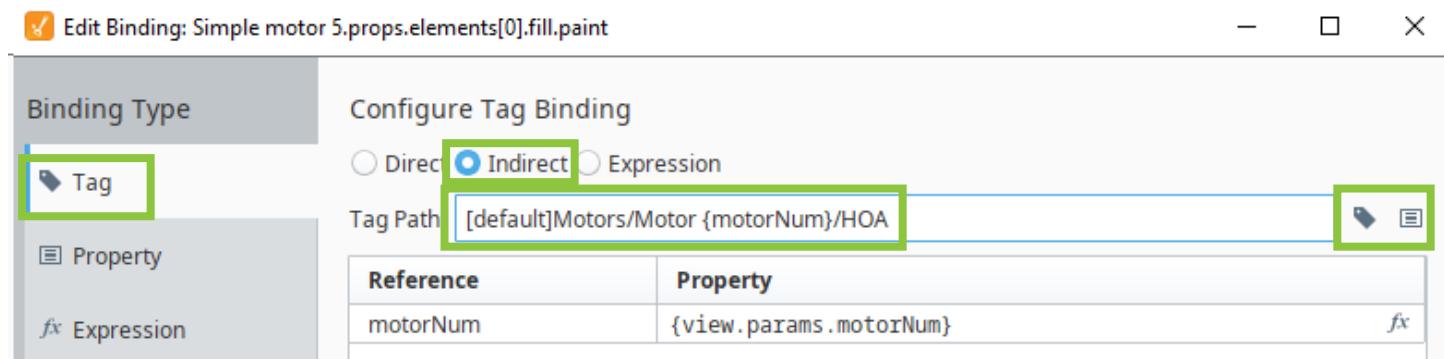
We've mentioned previously that Perspective properties are stored using JSON. Any time a property has nested sub-properties, they are represented as an array, which is the equivalent of a list in Python, or as an object, which is like a list but has named values, rather than values represented by index numbers. You will see examples of both in the following steps.

5. Expand **0**.
6. Expand **fill**.
7. Click  for the **paint** property.

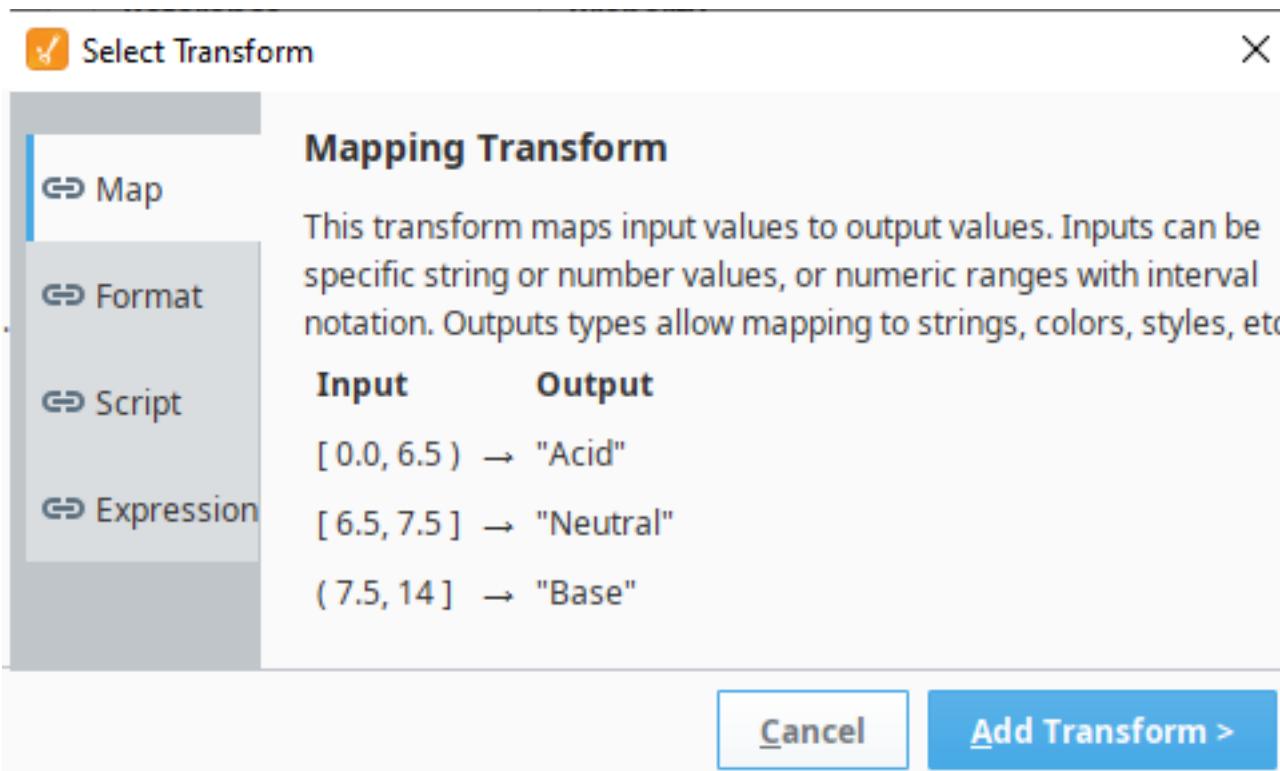


8. Click **Tag**.
9. Click **Indirect**.

10. Click .
11. Expand **Motors**.
12. Expand **Motor 1**.
13. Click **HOA**.
14. Click **OK**.
15. Delete the **1** in the Tag Path.
16. Click .
17. Expand **View**.
18. Expand **params**.
19. Click **motorNum**.
20. Click **OK**.



21. Click **Add Transform +**.
22. Click **Map**.
23. **Click Add Transform >**.



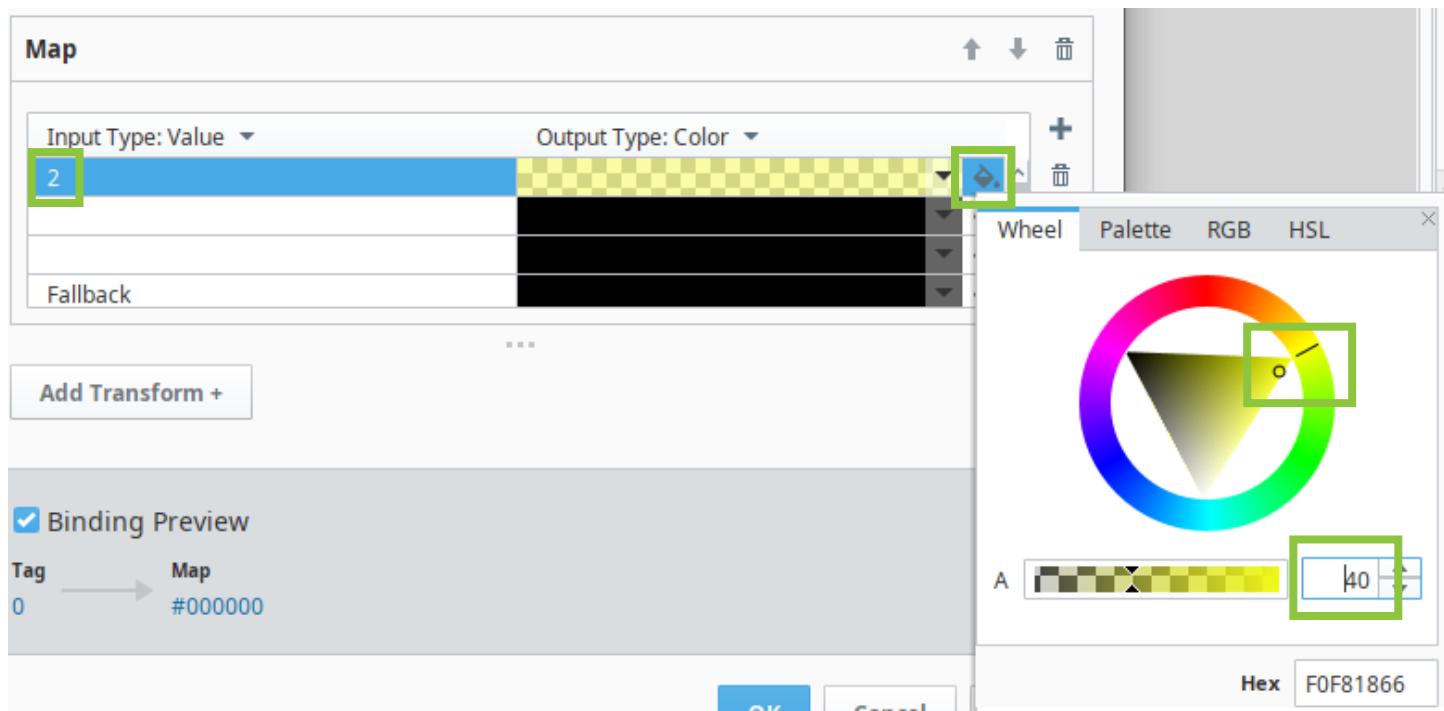
24. If necessary, make the dialog box taller.
25. Click **+** three times.
26. Select **Color** from the **Output Type** list.

Input Type: Value	Output Type: Value
	Value
	Color
	Style
	Style Class
	Expression
Fallback	Document

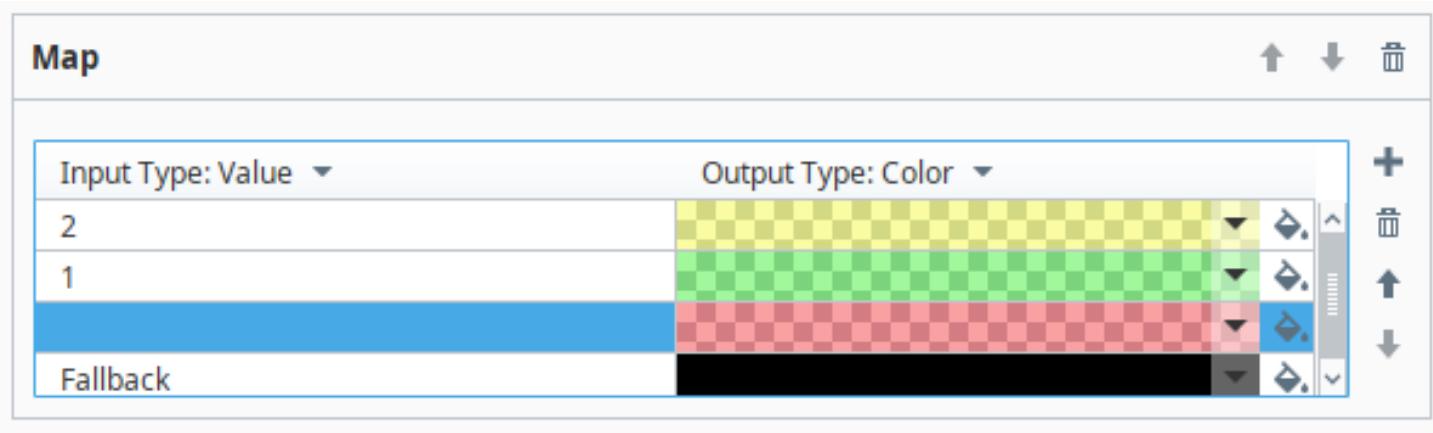
Add Transform +

27. Double-click in the **Input** column in the top row.
28. Enter **2**.

29. Click the **paint bucket** in the top cell of the Output column.
30. Click a **yellow color**.
31. Set **Alpha (transparency)** to **40**.



32. Double-click in the **Input** column in the second row.
33. Enter **1**.
34. Click the **paint bucket** in the second cell of the Output column.
35. Click a **green** color.
36. Set **Alpha (transparency)** to **40**.
37. Double-click in the **Input** column in the third row.
38. Enter **0**.
39. Click the **paint bucket** in the third cell of the Output column.
40. Click a **red color**.
41. Set **Alpha (transparency)** to **40**.
42. Click **OK**.



Resizing the Coordinate Container

If we resize the MotorView, we will see that the components inside Flex containers resize based on the properties we set. The motor image, however, is in a Coordinate container with a fixed size, so it does not resize. We can fix that, however, by changing the mode property of the Coordinate container.

1. Click the nested **Coordinate container**.
2. Select **percent** from the **mode** list.
3. Click **MotorView**.
4. Resize the **View**.

Embedding Views

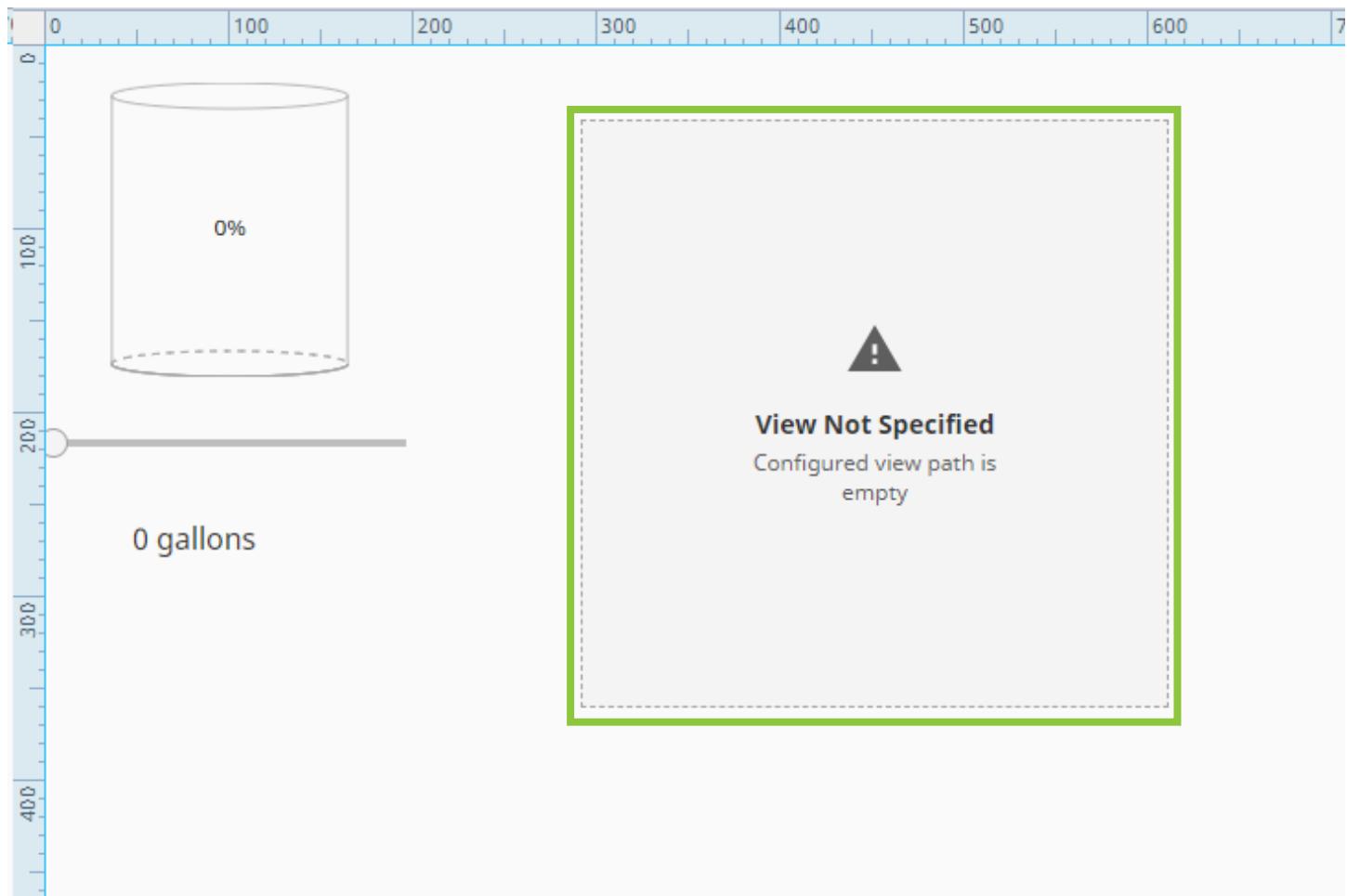
Now that we have a reusable View, we can add it to any other Views in the project. To add it to a larger View, we need to embed it. Perspective provides a variety of ways to embed Views, but we will examine only two: the Embedded View component, and the Carousel component.

Embedded View Component

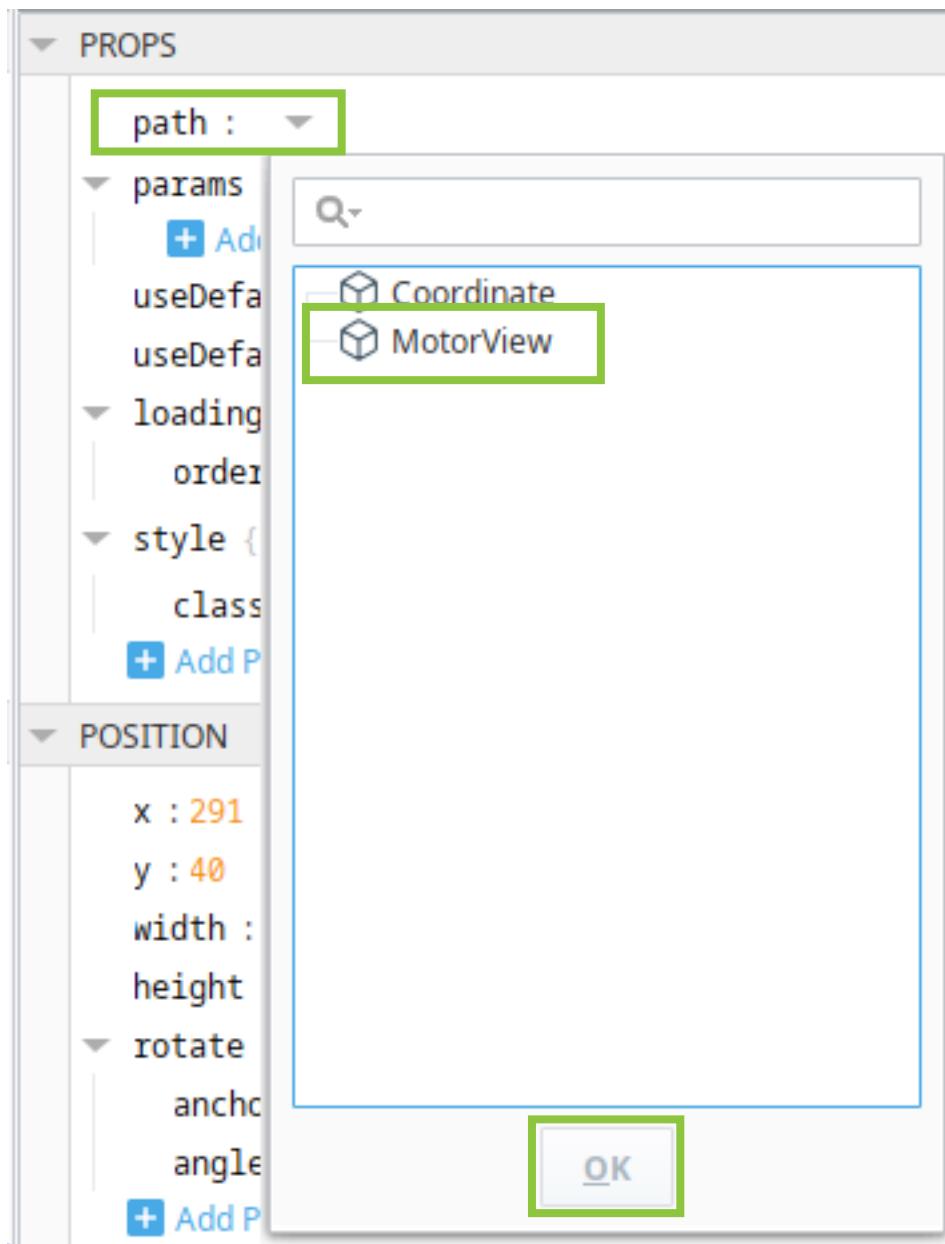
The Embedded View component most closely resembles the way to added Templates to Windows in Vision. The Embedded View component allows you to select a single other View and display it. Of course, you can have multiple Embedded View components on a View to display multiple other Views, or the same View multiple times.

1. Open the **Coordinate** View.

2. Click **Perspective Components**.
3. Enter **embed** in the Search box.
4. Drag an **Embedded View** to the View.



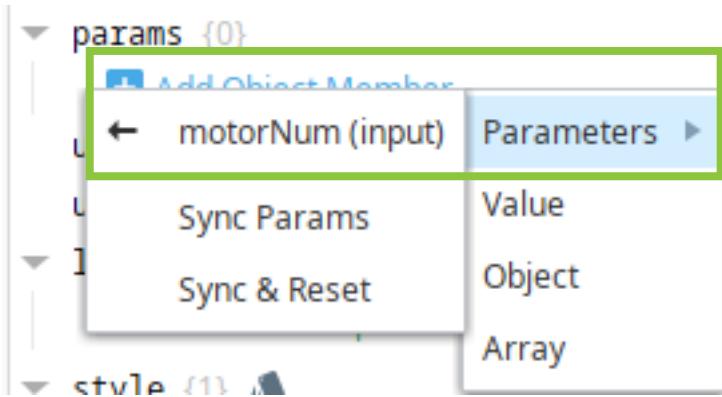
5. Select **MotorView** from the **View Path** property list.
6. Click **OK**.



Adding a Parameter

In order to get the Embedded View to display the motor that we want, we need to add a parameter to the component.

7. Click **Add Object Member** under the params property.
8. Select **parameters**.
9. Select **motorNum**.



Duplicate the View

Now that we have the Embedded View set up to display Motor 1, we can duplicate the component to display another motor.

10. Press **Ctrl** on your keyboard.
11. Drag the **Embedded View** component.
12. Enter a different value in the **motorNum** property.

Carousel Component

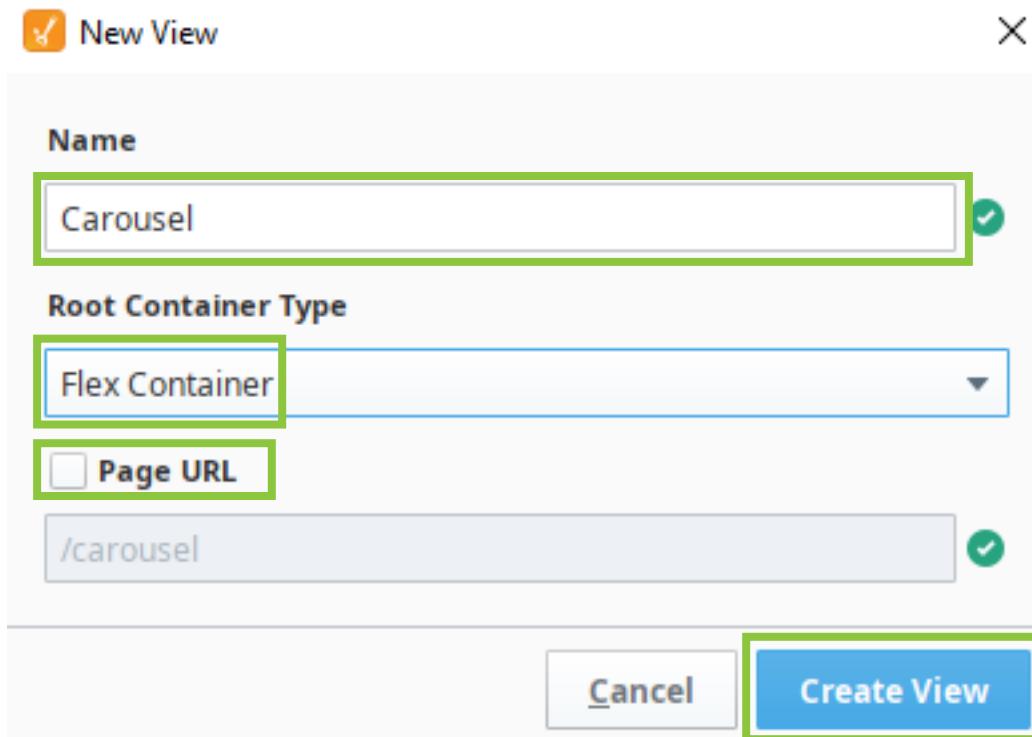
Another way to display multiple Views on other Views is to use the Carousel component. You have probably seen carousels on web sites before. In fact, inductiveautomation.com has one on our home page. (Our website is not built in Perspective, but because Perspective is just building web pages, we can see similarities between what we do on our site and what can be done in Perspective.) Carousels allow web sites to “stack” multiple pieces of content in one place, and then have the content scroll in and out ... as if it were going around a carousel. The Perspective component does the same: it allows us to stack multiple copies of a View (or a component), or even multiple different Views, on top of each other, and then scroll through them, one at a time.

While a Carousel could be used on any View, we are going to create a new View that will only contain the Carousel.

1. Click **Perspective** in the Project Browser.
2. Click **Create New View**.
3. Enter **Carousel** in the name field.
4. Select **Flex Container** from the Root Container Type list.

5. Leave **Page URL** unchecked.

6. Click **Create View**.



7. Click **Perspective Components**.

8. Enter **carousel** in the Search box.

9. Drag a **Carousel** to the View.

10. Enter 1 for the grow property.

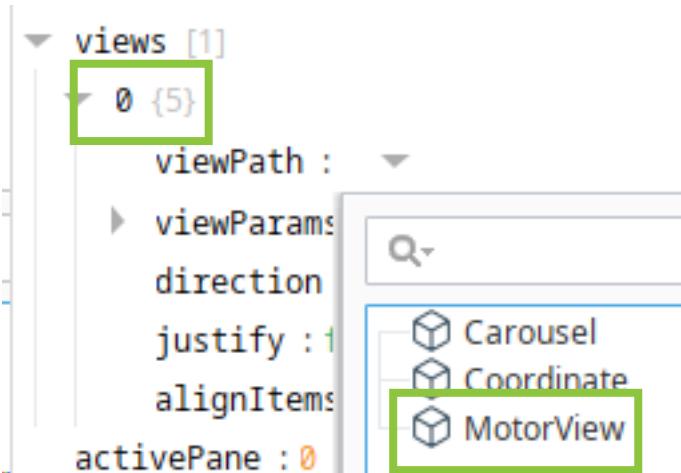
Like the Embedded View, the Carousel has a ViewPath property to set the View to be displayed. However, the Carousel's ViewPath is part of an array, as the component assumes we will have more than one View being displayed in it.

11. Expand **Views**.

12. Expand **O**.

13. Select **MotorView** from the **ViewPath** list.

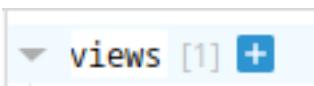
14. Click **OK**.



15. Expand **viewParams**.
16. Click **Add Object Member**.
17. Select **Parameters**.
18. Select **motorNum**.
19. Enter a **value**.

We can add additional Views to the Carousel by adding elements to the View array.

20. Mouse over **views**.
21. Click

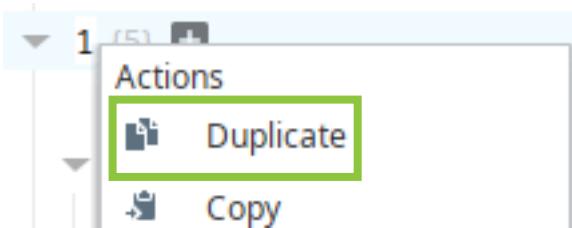


A new element is added to the array, so now we can configure its parameters.

1. Expand **1**.
2. Select **MotorView** from the **viewPath** list.
3. Expand **viewParams**.
4. Click **Add Object Member**.
5. Select **Parameters**.
6. Select **motorNum**.
7. Enter a **value**.

That works, but it is a hassle to have to set the View and add the parameters each time. Let's try a different approach. Instead of adding a new member to the array, we can duplicate one we already have.

8. Right-click **1**.
9. Click **Duplicate**.



You will see that we now have a new element in the array, 2. It already have the ViewPath assigned and the motorNum parameter added, so we can simply change its value to another motor.

You can in theory add as many elements to the Carousel as you want, but we are going to stop at 3.

Breakpoint Layout

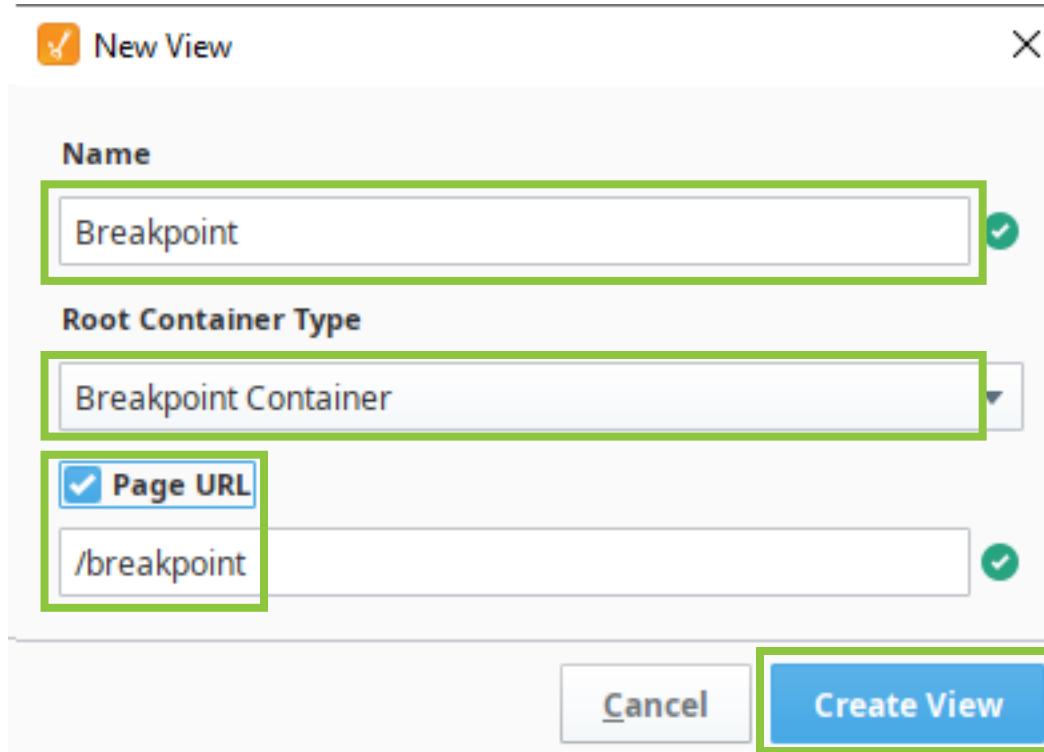
The biggest challenge modern web designers face is the need to create sites that will display properly on any size screen, from tiny mobile devices to huge billboards, and everything in between. We have already discussed the idea of responsive design and have seen how Flex containers can help.

Responsive design is based around a concept known as “breakpoints:” designers create layouts to fit one screen, and then as that screen gets bigger or smaller, there will be points where the layout breaks. At that point, there are several techniques to fix the layout to fit the new screen, and in Perspective, we have a container specifically designed for this purpose: the aptly named Breakpoint container.

The Breakpoint container allows you to embed two other Views: one for large screens, and a second for small screens. We have created two Views for this purpose: we will use the Coordinate View for the large breakpoint, and the Carousel for the small.

1. Click **Perspective** in the Project Browser.
2. Click **Create New View**.
3. Enter **Breakpoint** in the name field.

4. Select **Breakpoint Container** from the Root Container Enter list.
5. Click **Page URL**. We'll accept the default URL.
6. Click **Create View**.



Look at the top of the Perspective Property Editor. The Breakpoint container has three key properties: the breakpoint, and large and small children. The Breakpoint is where you determine the size at which the container will switch between the large and small Views. For our purposes, we are going to leave the Breakpoint at its default 640px, but for most layouts, this will be too small and will need to be changed.

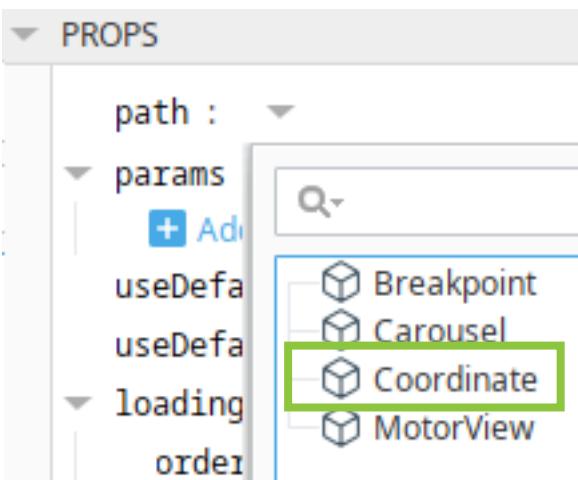
Before you embed a View, you need to be sure that you have the correct Child selected. This is the only visual indicator as to which Child you are embedding the View into, so it's important to pay close attention to your selection.

Each Child View can only contain a single component, so most often, you will place an embedded View

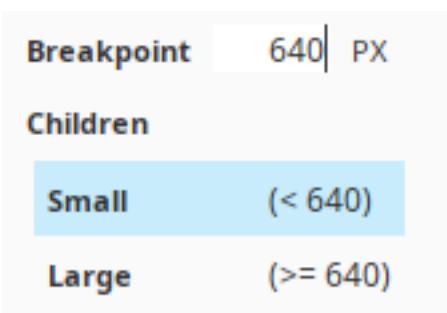
1. Click **Large**.



2. Click **Perspective Components**.
3. Enter **embed** in the Search box.
4. Drag an **Embedded View** component to the View.
5. Select **Coordinate** from the **path** list.
6. Click **OK**.



7. Click **Small**.

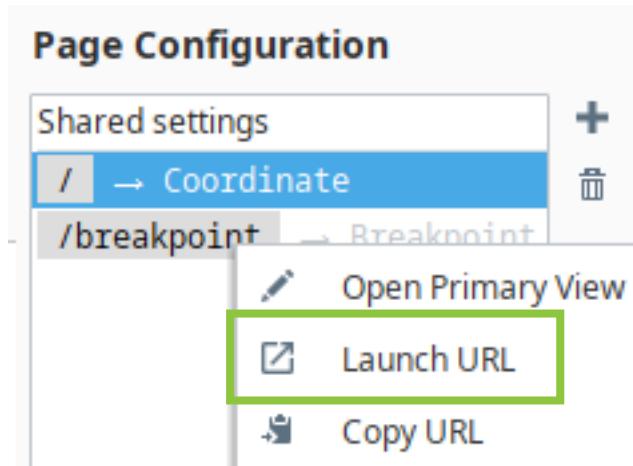


8. Click **Perspective Components**.
9. Enter **embed** in the Search box.
10. Drag an **Embedded View** component to the View.
11. Select **Carousel** from the **Path** list.
12. Click **OK**.
13. Click **File**.
14. Click **Save All**.

Preview a Specific View

You can preview a specific View in the browser without needing to create navigation.

1. Click **Perspective** in the Project Browser.
2. Right-click **Breakpoint**.
3. Click **Launch URL**.



4. Change the size of your browser to see your breakpoint change

Styles

As noted above, all formatting on the web is done using Cascading Style Sheets, or CSS. Because Perspective is ultimately creating web sites, formatting in Perspective is also done with CSS.

CSS is designed around the idea of separating content from presentation. In Perspective, the content is our components. We need to create styles to define our formatting, and then we can apply those styles to components. Style definitions in Perspective are called classes.

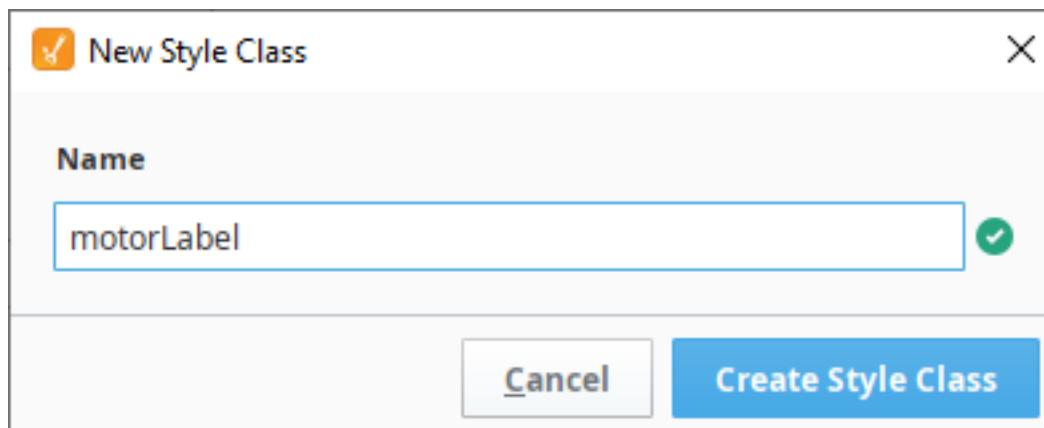
Each class can be applied to any number of components. This dramatically increases the maintainability of your project, as you can redefine a class at any point and every component that class is applied to will automatically update.

You should name classes based on how they will be used rather than how they look. So, **motorLabel** is a better name than **redBoldLabel**.

Creating a Class

We create classes from the **Styles** object in the Project Browser.

1. Right-click **Styles** on the Project Browser.
2. Click **New Style**.
3. Enter **motorLabel**.
4. Click **Create Style Class**.

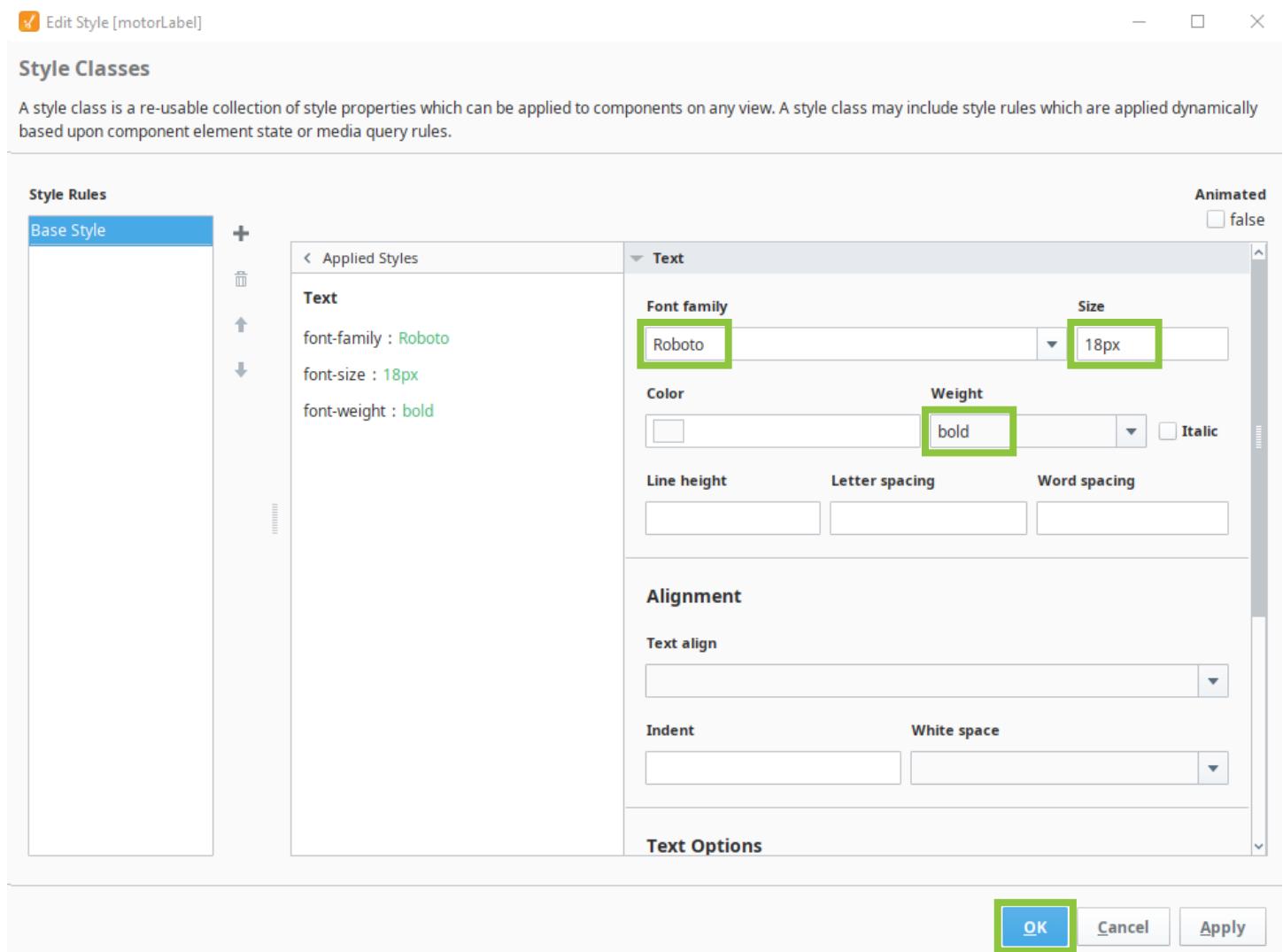


There are many properties available in CSS, and only a small subset of them are available in the Edit Style dialog box. For our purposes, we are only going to focus on a few properties, mostly relating to text formatting.

5. Click **Text**.
6. Select a font from the **Font Family** list.
7. Enter **18px** in the **Size** field.

Note: There is not default unit of measurement in CSS, so you must always specify the units you are using. Pixels (px) and percents (%) are the most common and easiest to understand units. There are many resources available online to describe CSS's other units.

8. Select **bold** from the Weight list.
9. Click **OK**.

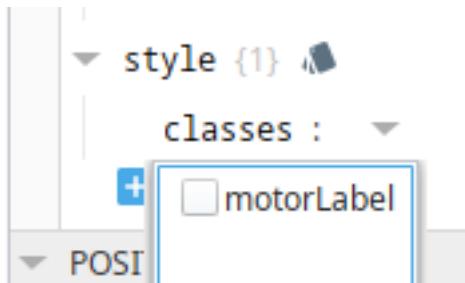


Applying a Class

Now that we have a style class defined, we can apply to it components in our Views.

10. Double-click **MotorView**.
11. Double-click the nested **Flex** container.
12. Click the **Label**.

13. Select **motorLabel** from the classes list.



The label will update to display with the properties from the style.

Creating Another Class

Let's create a second class, to style the header in our MotorView.

1. Right-click **Styles** on the Project Browser.
2. Click **New Style**.
3. Enter **motorHeader**.
4. Click **Create Style Class**.
5. Click Text.
6. Select a font from the **Font Family** list.
7. Enter **36px** in the Size field.
8. Select **bold** from the Weight list.
9. Click **OK**.
10. Double-click **MotorView**.
11. Click the **Label** at the top of the View.
12. Select **motorHeader** from the classes list.



Edit a Class

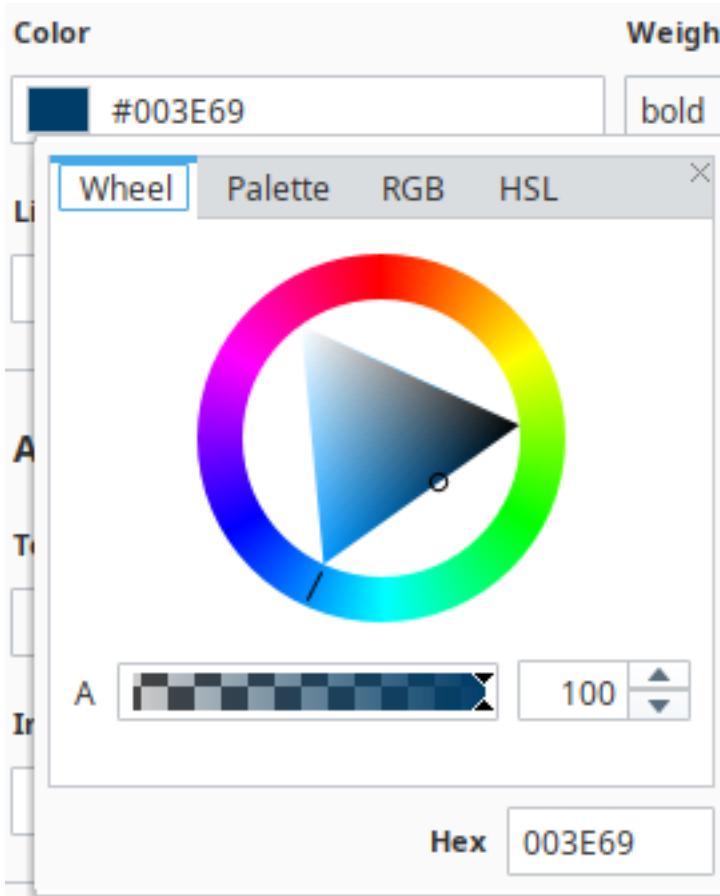
We can return to a class to edit it at any time.

We want to edit the class for our header to change its appearance when the user moves their mouse over the header.

13. Double-click **motorHeader**.
14. Click **Text**.

15. Click the white box in the **Color** field.

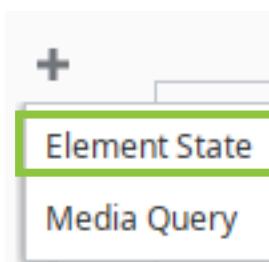
16. Click a **dark blue color**.



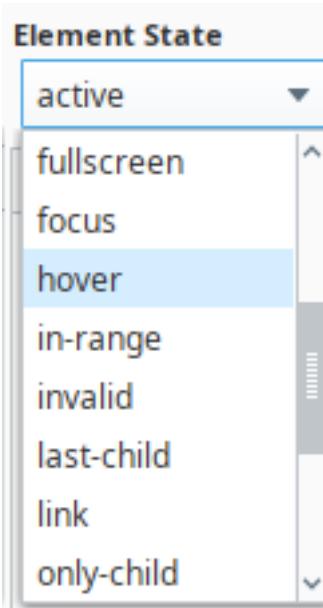
To get a mouseover effect, we need to add an element state to the class. This will define a set of style properties that we can have the browser automatically apply when a state is met.

17. Click **+**.

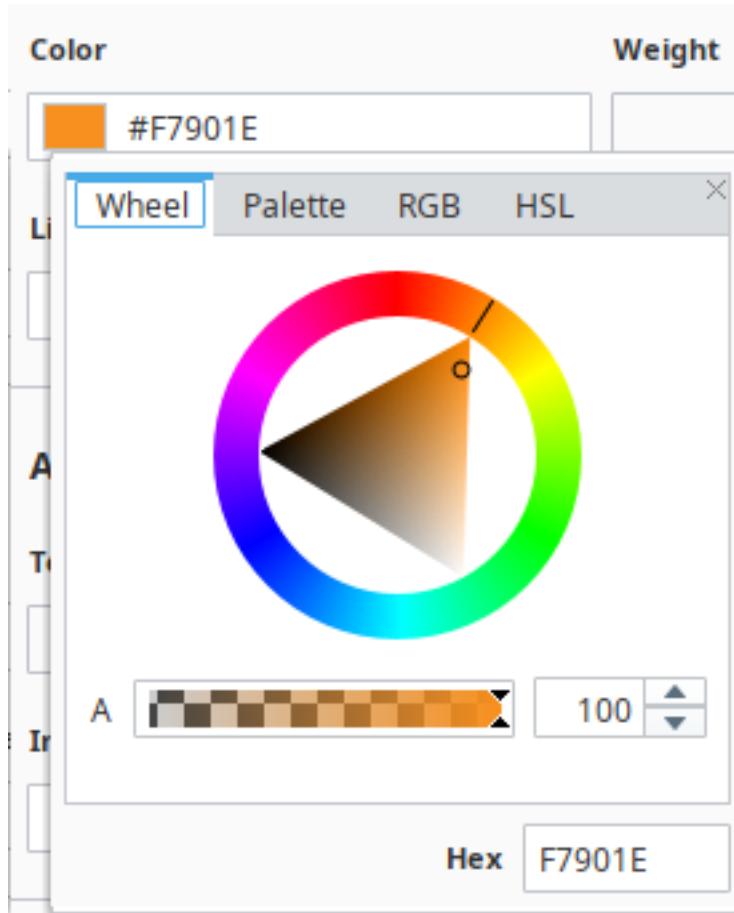
18. Select **Element State**.



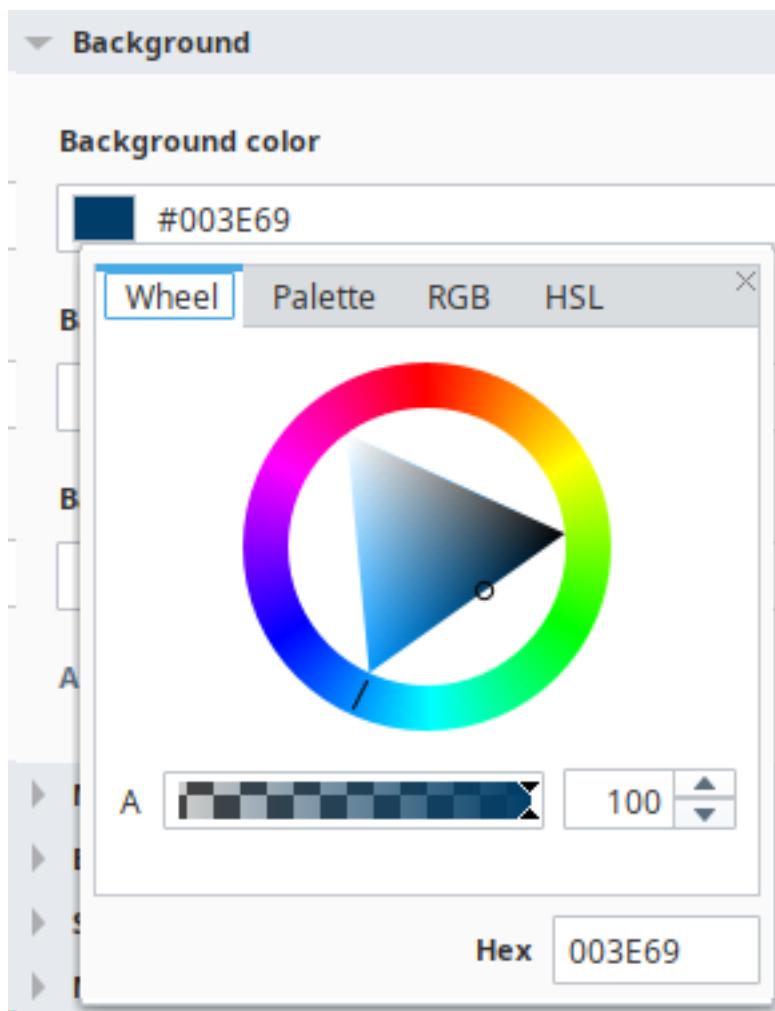
19. Select **hover** from the Element State list.



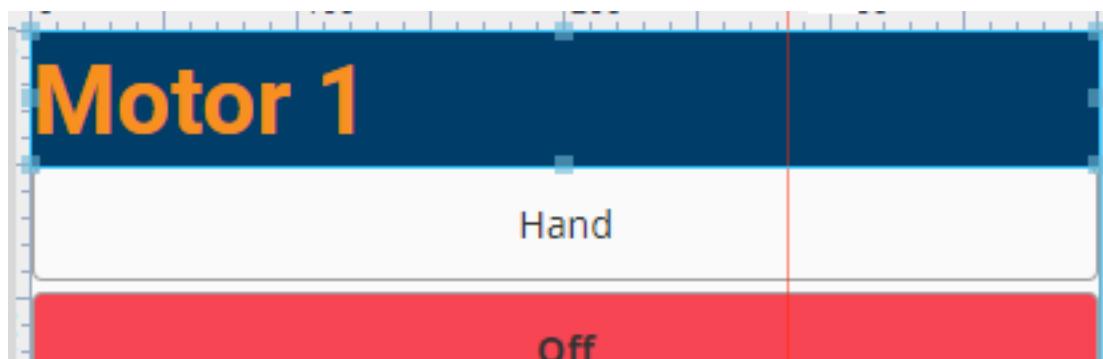
20. Click **Text**.
21. Click the white box in the **Color** field.
22. Click a **bright color**.



23. Click **Background**. You may need to scroll down.
24. Click the white box in the **Background** color field.
25. Click a **dark blue** color.



26. Click **OK**.
27. Return to the **MotorView**.
28. Move your mouse over the header.



Element states apply automatically and can be preViewed even in Design mode.

Best Practice: Always Use Classes

It is possible to style components directly by applying CSS properties via the menu next to Styles on the Perspective Property Editor. However, this practice is not recommended. The real power of CSS is its separation of formatting from content. By applying your formatting exclusively through classes, you can update the visual look and feel of your project by merely editing the classes. If you apply styles directly, however, you will have to open each individual View, find each component, and change its styles. In addition, if you apply styles directly to components and then apply a class to the same component, the locally-applied styles will override the class wherever they conflict (this is the “cascading” part of CSS’s name.)

Perspective Security

With Perspective running in the web browser, security is handled differently than it is in Vision Clients. Identity Providers are used to authenticate users in a Perspective Session. An Identity Provider (IdP) is a system that maintains user information and can authenticate users across multiple applications. There are two main systems that IdPs use: OpenID and SAML. Each IdP uses one of those systems and Ignition can connect to either type.

While Ignition can be connected to third party IdPs, Ignition’s User Sources can also be used as an IdP. If you go into the Identity Providers page in the Gateway Webpage, you will notice that an IdP already exists called default. This IdP is linked to the User Source called default and will authenticate all the users in that User Source.

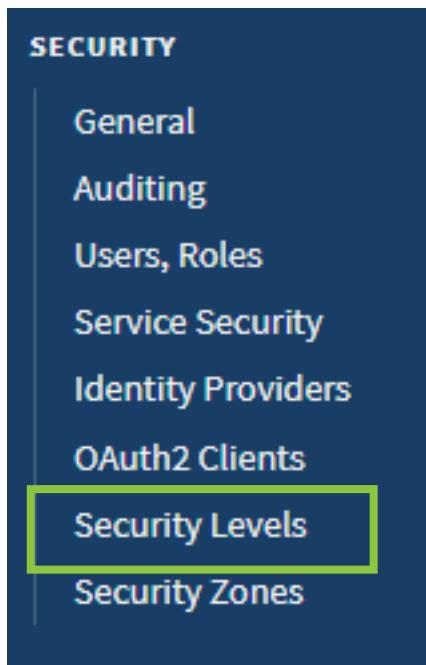
Security Levels

Security Levels are a new security system that works with the user information provided by the IdP to grant users certain permissions that can then later be applied to a Perspective project. One advanTage of Security Levels is that they are not solely roles-based and can instead utilize other information to determine what permissions a user should have.

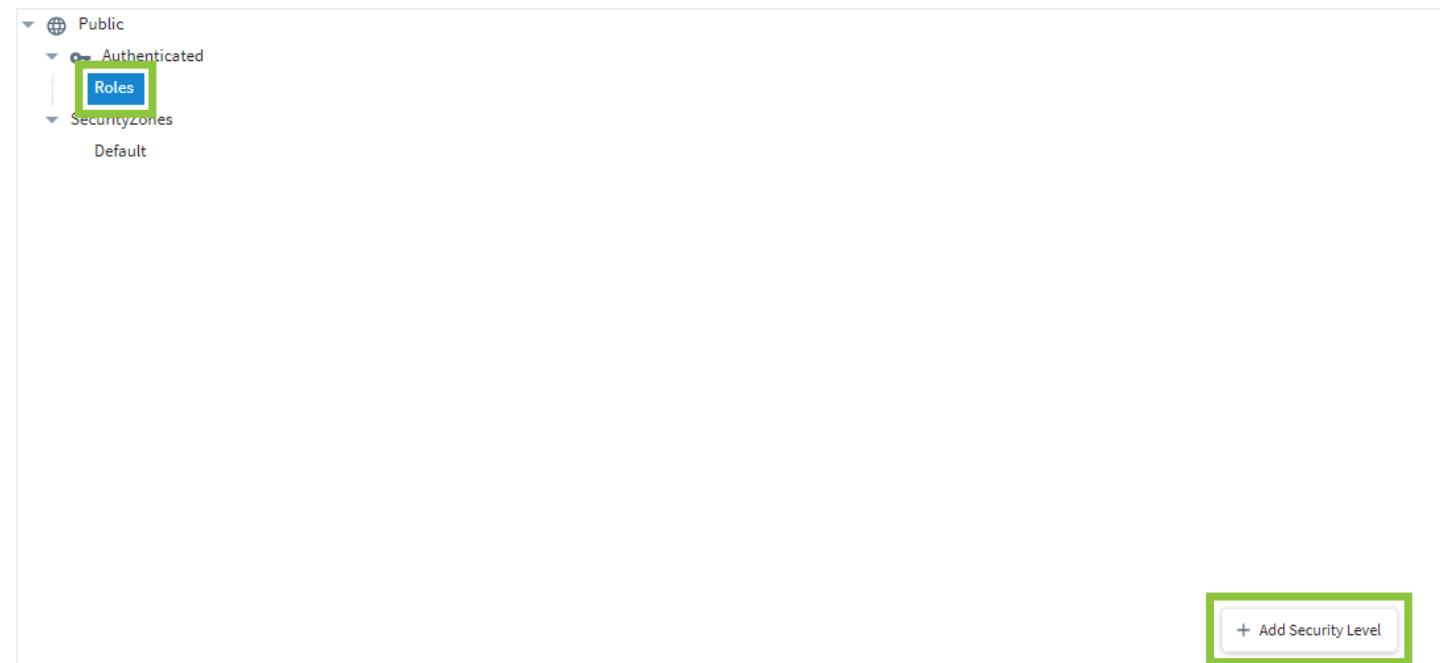
We will begin to explore Perspective Security by creating our own Security Levels to match the roles we have in our user source.

1. Open the **Gateway** webpage.

2. **Log in.**
3. Click **Config**.
4. Click **Security Levels** under the **Security** heading.



5. Click **Roles**.
6. Click **+ Add Security Level**.



7. Enter **Administrator** in the Name field.

Security Level Details

Name

This screenshot shows a software interface for creating a new security level. The title 'Security Level Details' is at the top. Below it is a 'Name' label followed by a text input field containing the word 'Administrator'. The entire form is enclosed in a light gray box.

8. Click **Roles**.
9. Click **+ Add Security Level**.
10. Enter **Operator** in the Name field.
11. Click **Roles**.
12. Click **+ Add Security Level**.
13. Enter **Manager** in the Name field.
14. Click **Save**.

The screenshot shows the Perspective software's security configuration screen. On the left, there's a tree view of security levels:

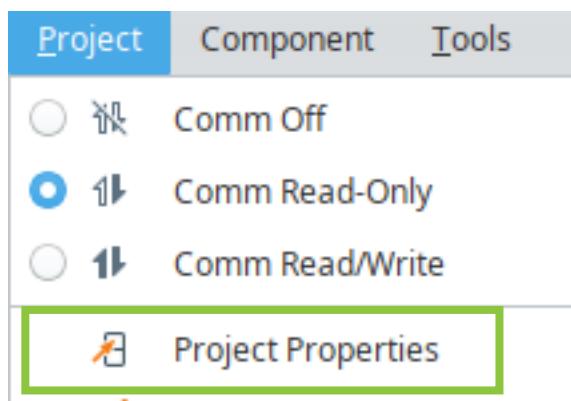
- Public
- Authenticated
 - Roles
 - Administrator
 - Manager**
 - Operator
 - SecurityZones
 - Default

In the bottom right corner, there is a blue "Save" button.

Security in the Perspective Project

Now that we have Security Levels set up, we can use them within the project.

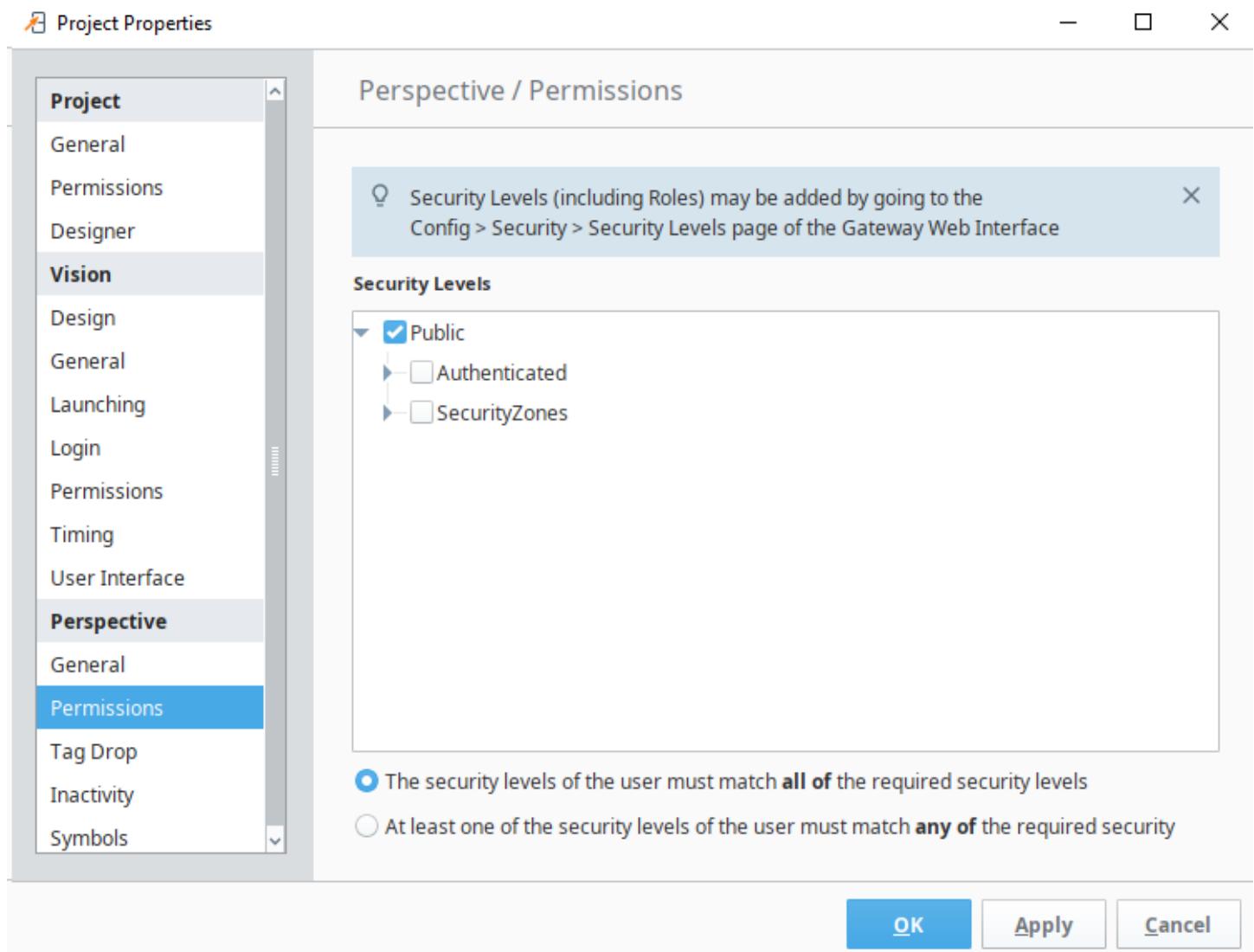
15. Return to **Designer**.
16. Click **Project**.
17. Click **Project Properties**.



18. Click **Permissions** in the Perspective section.

19. Review the security options.

20. Click **OK**.



You will see that by default, security on the project is set to Public, meaning anyone with the URL to the project can access it. We already saw this when we preViewed the project earlier and did not have to log in.

We have several options to secure our project.

Authenticated will require a log in, but anyone with an account in our IdP will be able to log in, including our Guest role.

But if you expand Authenticated, you will see an option for Roles. Checking this will also require a log in but will restrict access to those accounts that are in Roles. In our project, this would have the affect of locking out our Guest user.

We can also expand Roles and select one or more roles, which would limit access to only users in those roles. However, it is important to note the option at the bottom of the screen: by default, if you select multiple roles, only users in all those roles will have access. To enable access for users in, for example, either the Administrator or the Manager role, you would need to select those roles and be sure to select the second option at the bottom.

Securing a View

Each View can also be secured using Security Levels. Simply right-click on an open View and select Configure View Permissions. Here you can configure permissions for a View much like you would for the whole project.

Perspective Navigation

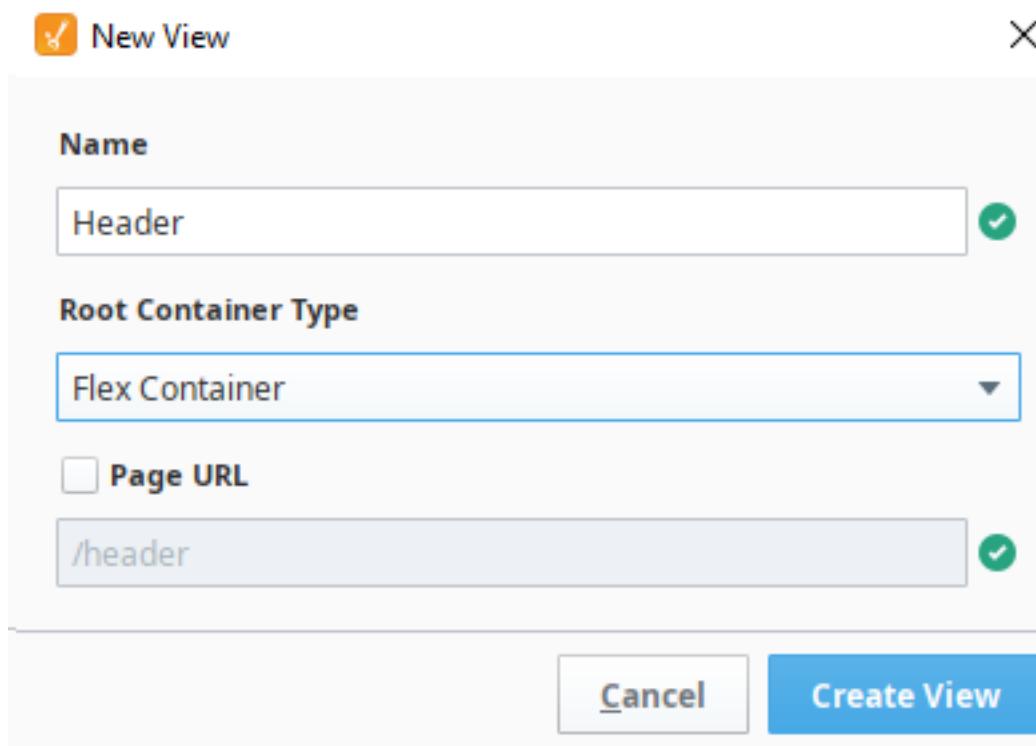
In Vision, you cannot access a Window in the Client unless you specifically create some way to navigate to it. In Perspective, however, you can navigate to any page (in other words, any View with a Page URL) by simply typing the appropriate URL into the browser's address bar.

That said, it is obviously preferable to provide some kind of navigation in your projects.

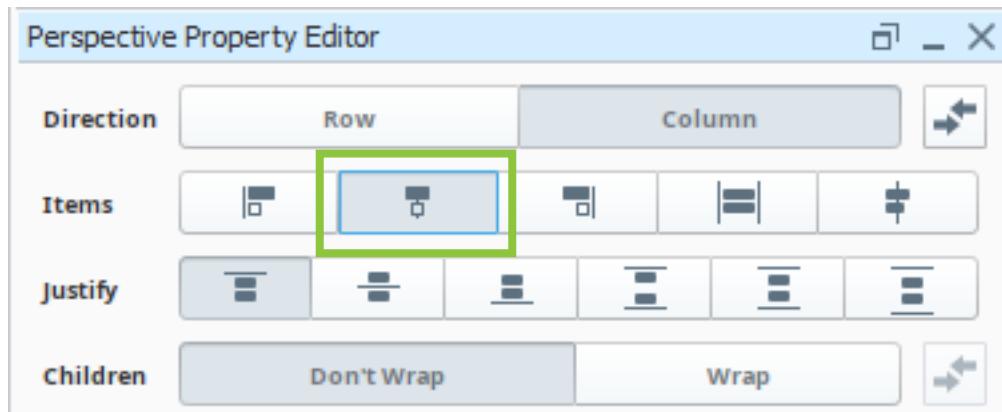
Create a Header View

We will want to add our navigation in a way that will ensure that it is visible on all pages in our project, including pages that we create later. To do this, we will want to add a new View that we can then dock to the top of every page, in much the same way that we had a Navigation Window docked in our Vision Client.

1. Click **Perspective** in the Project Browser.
2. Click **Create New View**.
3. Enter **Header** in the Name field.
4. Select **Flex Container** from the Root Container Enter list.
5. Leave **Page URL** unchecked.
6. Click **Create View**.

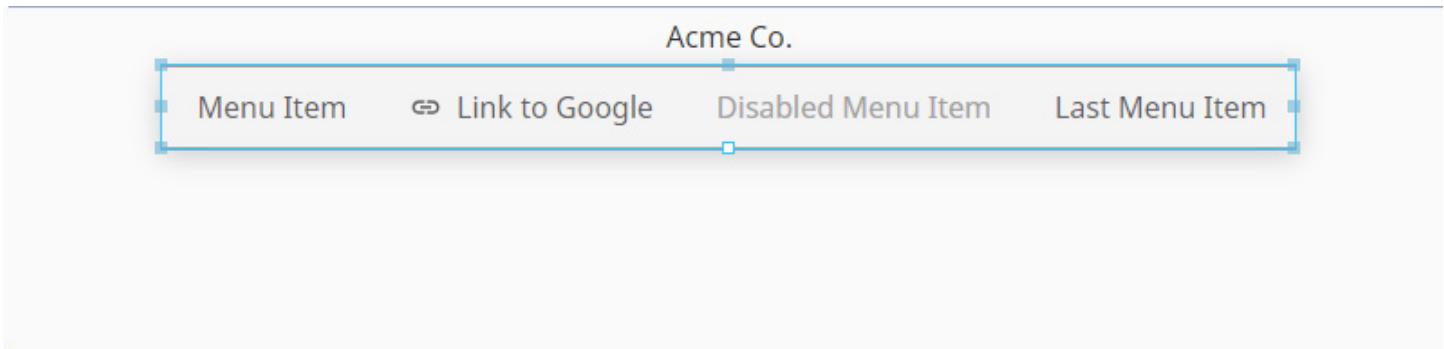


7. Resize the View to be approximately **200 pixels** tall.
8. Click **Perspective Components**.
9. Enter **label** in the Search box.
10. Drag a **Label** to the View.
11. Enter **Acme Co.** in the text property.
12. Click **Center** in the Items section of the Perspective Property Editor.



Add a Horizontal Menu

1. Click **Perspective Components**.
2. Enter **Menu** in the Search box.
3. Drag a **Horizontal Menu** to the View.



The Horizontal Menu component is a navigation system for Perspective that comes preset with examples of the things you might want on a menu. The first item contains a nested menu with a link to an external site. The second is a link to an external site with an icon. The third shows that items can be disabled, and then final item is a normal link to a page inside our project. All of these are set by properties in the items array, so we can alter this menu to serve our needs by changing those items.

Edit the Menu

First, we want to remove the nested menu and have the first item simply point to our home page.

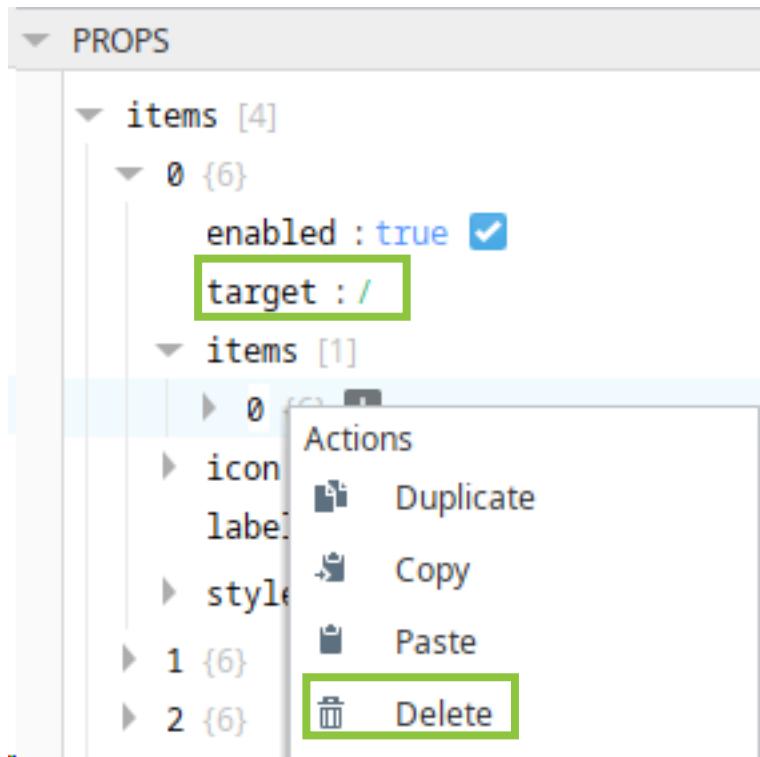
1. Expand **0** under the items array.
2. Enter **/** in the target field.

The target property sets the URL that the user will be taken to when clicking this link. Unfortunately, all URLs, including those within our project, must be manually typed in this field.

This menu has a nested item because its items property has an array element in it. We need to delete that to remove the nested item.

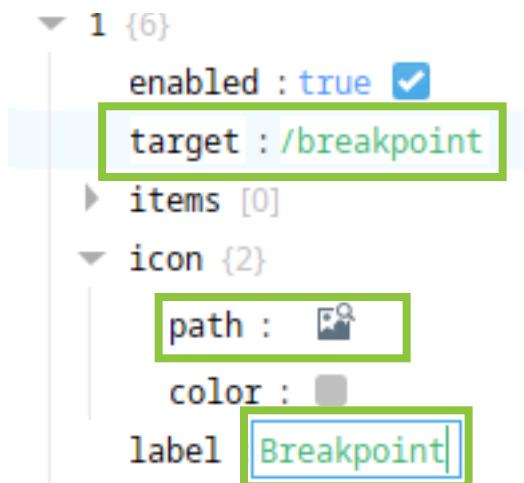
3. Expand **items**.
4. Right-click **0**.
5. Click **Delete**.

6. Enter **Home** in the label field.



Now, we can modify the second item to remove the icon and provide a link to another page within our project.

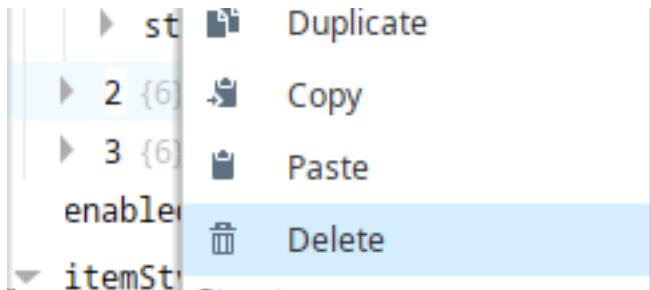
7. Expand **1**.
8. Enter **/breakpoint** in the target field.
9. Expand **icon**.
10. Delete the **text** in the path field.
11. Enter **Breakpoint** in the label field.



Right now, those are the only two pages we have in our project, so we'll delete the other two items in the menu.

12. Right-click **2**.

13. Click **Delete**.



14. Right-click **3**.

15. Click **Delete**.

If you decide later you need more pages in your menu, you can simply add new items by clicking the + next to items, or you could duplicate an existing item in the array.

Style the Header Text

Next, we will create a style to format the heading text.

1. Right-click **Styles** in the Project Browser.

2. Click **New Style...**

3. Enter **header** in the Name field.

4. Click **Create Style Class**.

5. Click **Text**.

6. Select a font from the **Font family** list.

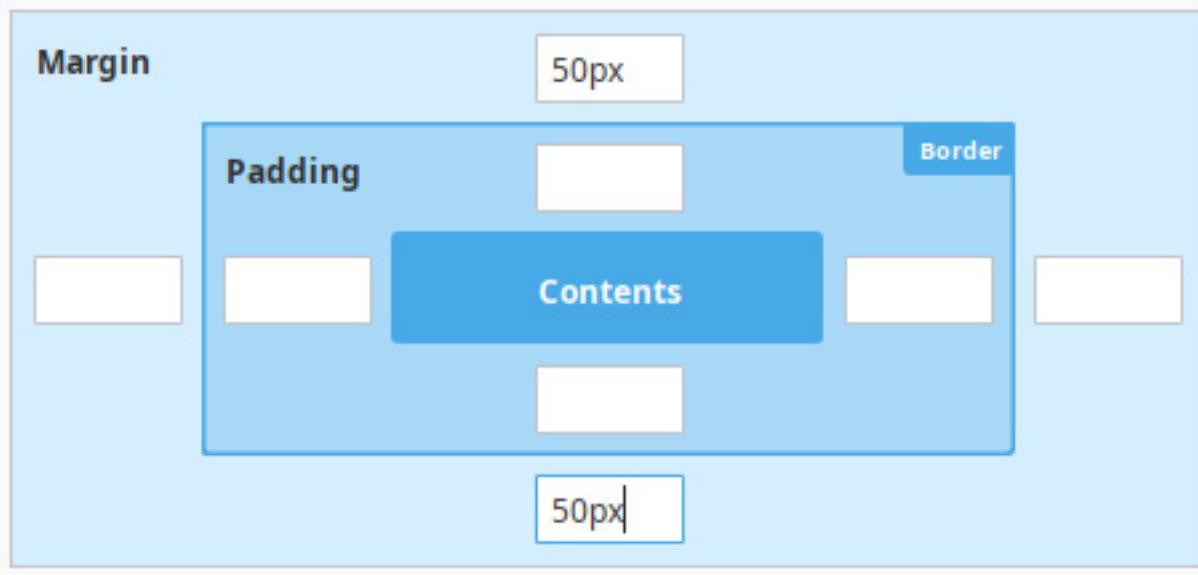
7. Enter **300%** in the Size field.

We also want to force some space around this text, which we can do by adding a margin to the header.

8. Click **Margin and Padding**.

9. Enter **50px** in the top margin field.

10. Enter **50px** in the bottom margin field.

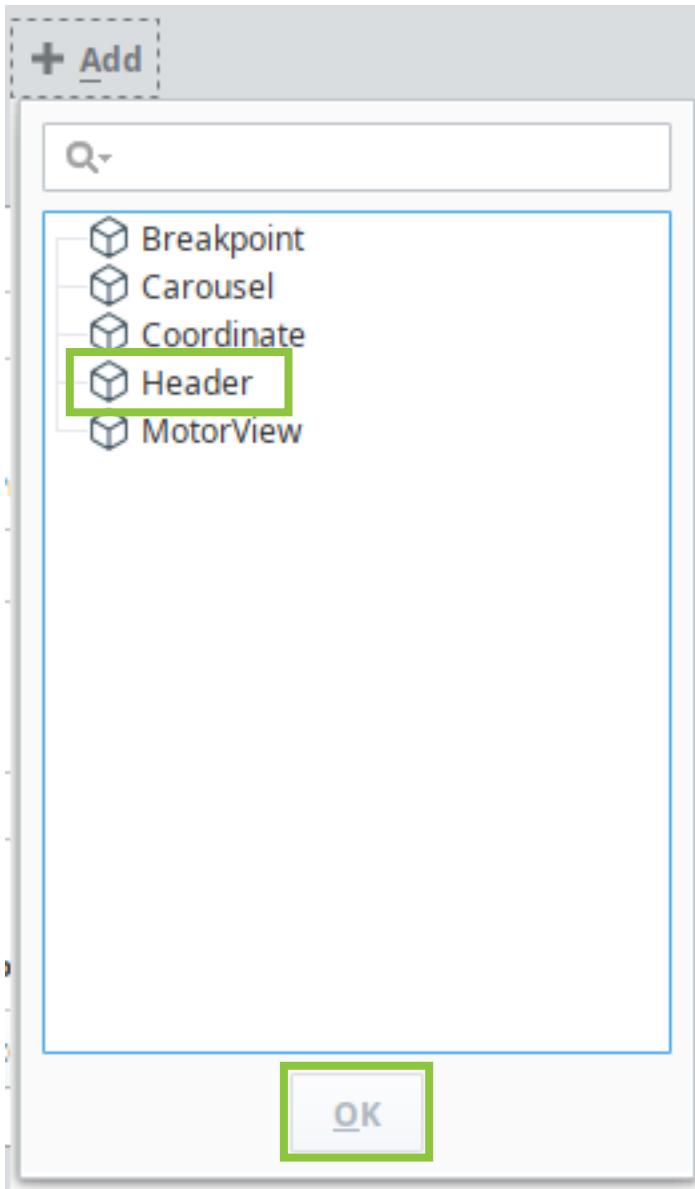


11. Click **OK**.
12. Click the **Label**.
13. Select **header** from the classes list.
14. Click **File**.
15. Click **Save All**.

Dock a View in Shared Settings

The final step is to make our header appear at the top of every page. We can do this by docking it in the Perspective Page Settings.

1. Click **Perspective** in the Project Browser.
2. Click **Shared Settings**.
3. Click **+ Add** in the top section.
4. Click **Header**.
5. Click **OK**.



6. Click **File**.
7. Click **Save All**.
8. Right-click **/**.
9. Click **Launch URL**.

The page will open in the browser, with the header docked at the top. You can click items in the navigation to move between pages.

Reporting

The Reporting module creates rich, dynamic PDF reports from any data. The Ignition Reporting Module is a standalone reporting solution that simplifies and enhances the reporting process from beginning to end. You can generate reports from existing files or create them from scratch. Pull data from a variety of sources, and design reports with a simple to use interface. Add charts, graphs, tables, crosstabs, shapes, and other components for visual impact. Save your reports in PDF, HTML, CSV, and RTF file formats, and set up automatic scheduling and delivery.

Creating a Basic PDF Report

The Reports Module lets you create dynamic PDF reports. Earlier in the class, we created a Transaction Group to store data from a certain set of Tags in the database. Now, we can use that data to create a report.

Verify the Data

Before we try to create the report, we want to verify that the Transaction Group is running and properly storing data.

1. Click **Transaction Group** in the Project Browser.
2. Click **History**.

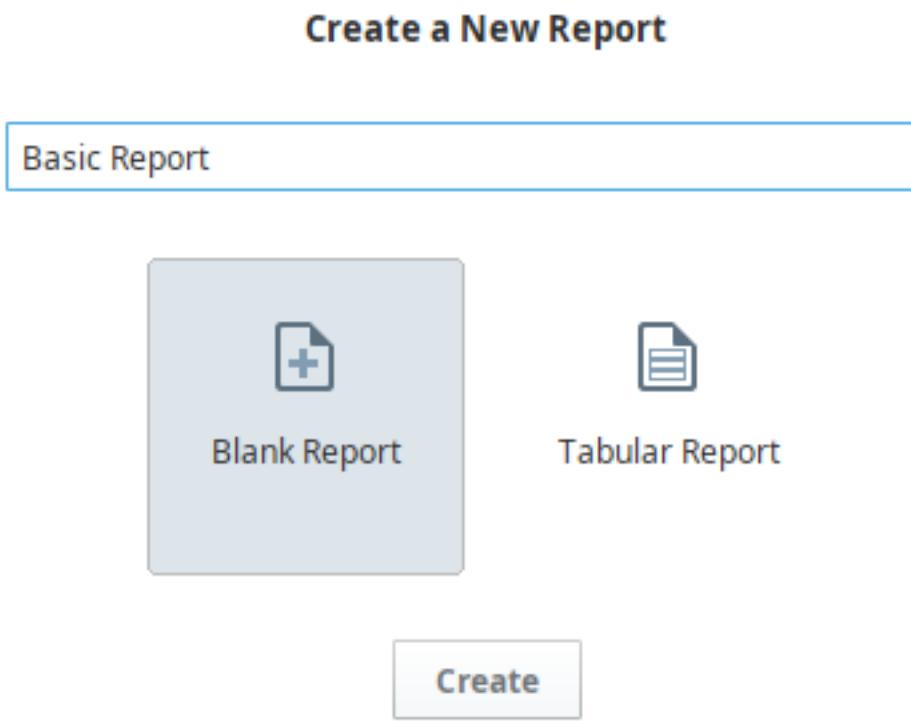
Ensure that the Transaction Group is running. If it isn't, click Enabled, then save the project.

If the Transaction Group is running but not storing data (make sure that the Total executions and DB writes fields are updating), check the trigger. If necessary, change the value of the TempF Tag in the Tag Browser so that the Transaction Group starts executing.

Create a Report

Now that we know we will have data, we need to switch to the Report module, and create a new report.

1. Click **Reports** in the Project Browser.
2. Enter **Basic Report** in the Name of the report field.
3. Ensure that **Blank Report** is selected.
4. Click **Create**.



Report Module Tabs

You will see five tabs at the top of the Report module.

Report Overview This tab is a collection of useful information about your report. It will be filled in as the report is generated and used.

Data This tab is where you define the data that goes into your report. You define the report's parameters here, as well as create the data sources that will be used for the report.

Design The primary workspace when creating a report. You will add components such as tables and charts here, as well as static text elements like headers and page numbers.

Preview Throughout the design process, you can switch to Preview to make sure the data is displaying correctly.

Schedule Set up schedules so your report can automatically be printed, saved, or emailed.

Adding Data to Your Report

Data for the report comes from two main sources: parameters you can pass in, and queries.

Report Parameters

Reports default to having two parameters, representing a date range for the report. You can see these in the top left corner of the Data tab.

The **StartDate** parameter uses an expression based on the dateArithmetic function. It takes the current timestamp—now()—and subtracts 8 hours from that.

Note: The expression functions do not contain methods to directly subtract values. Instead, all of the add functions allow you to add negative numbers for subtraction.

EndDate simply relies on now(). So, if used together, a default report will get the last 8 hours of data. But, remember that these are parameters, and whatever method you use to call the report can pass different values in for each. We will examine some of those possibilities as we move forward.

It is also possible to add additional parameters to your report if needed.

Report Data Sources

There are 8 main sources for the data in a report.

Named Query Named queries provide ways that you can save SQL statements into the project for reuse. They are beyond the scope of this course.

SQL Query This allows you to directly write SQL to query the database. We will use a SQL Query in this example.

Basic SQL Query The Basic SQL Query also allows you to directly write SQL to query the database. This is an older method for writing queries that is maintained for legacy use.

Tag Historian Query As was stated in Chapter 11, the Tag Historian is not designed to be queried directly. Instead, we can use the Tag Historian Query type to retrieve that data.

Tag Calculation Query This query also relies on the Tag Historian, but allows you to perform calculations such as sum, average, and the like. Using this data source is beyond the scope of this class.

Alarm Journal Query While the tables created for the Alarm Journal can be queried directly, this provides a simpler interface for working with that data. Using this data source is beyond the scope of this class.

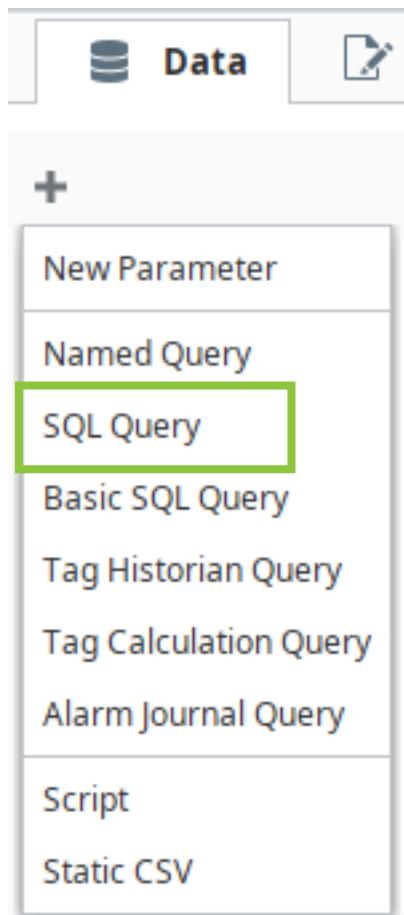
Script It's possible to write a Python script to retrieve data. This can be used if you need to get data from multiple sources or pre-process the data, but using this data source is beyond the scope of this class.

Static CSV Rather than retrieving data from a database, you can import a static CSV file, such as a file created in Excel, and use it for reports. Using this data source is beyond the scope of this class.

Create a Query

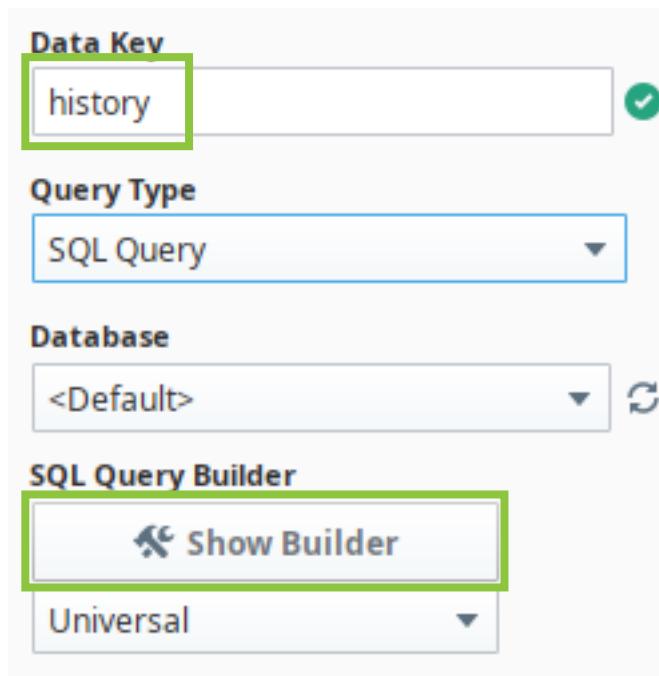
For our report, we want to use the SQL Query.

1. Click **Data**.
2. Click **+**.
3. Click **SQL Query**.



A certain level of knowledge of SQL is necessary to really create reports, but we can create a simple report with data from a single table—the one being used by our Transaction Group—with relative ease thanks to the SQL Query Builder tool, which allows us to create most of the query in an interactive visual interface. We also want to name our Data Key, which will become the name of the query for the rest of the report.

4. Enter **history** in the **Data Key** field.
5. Click **Show Builder**.

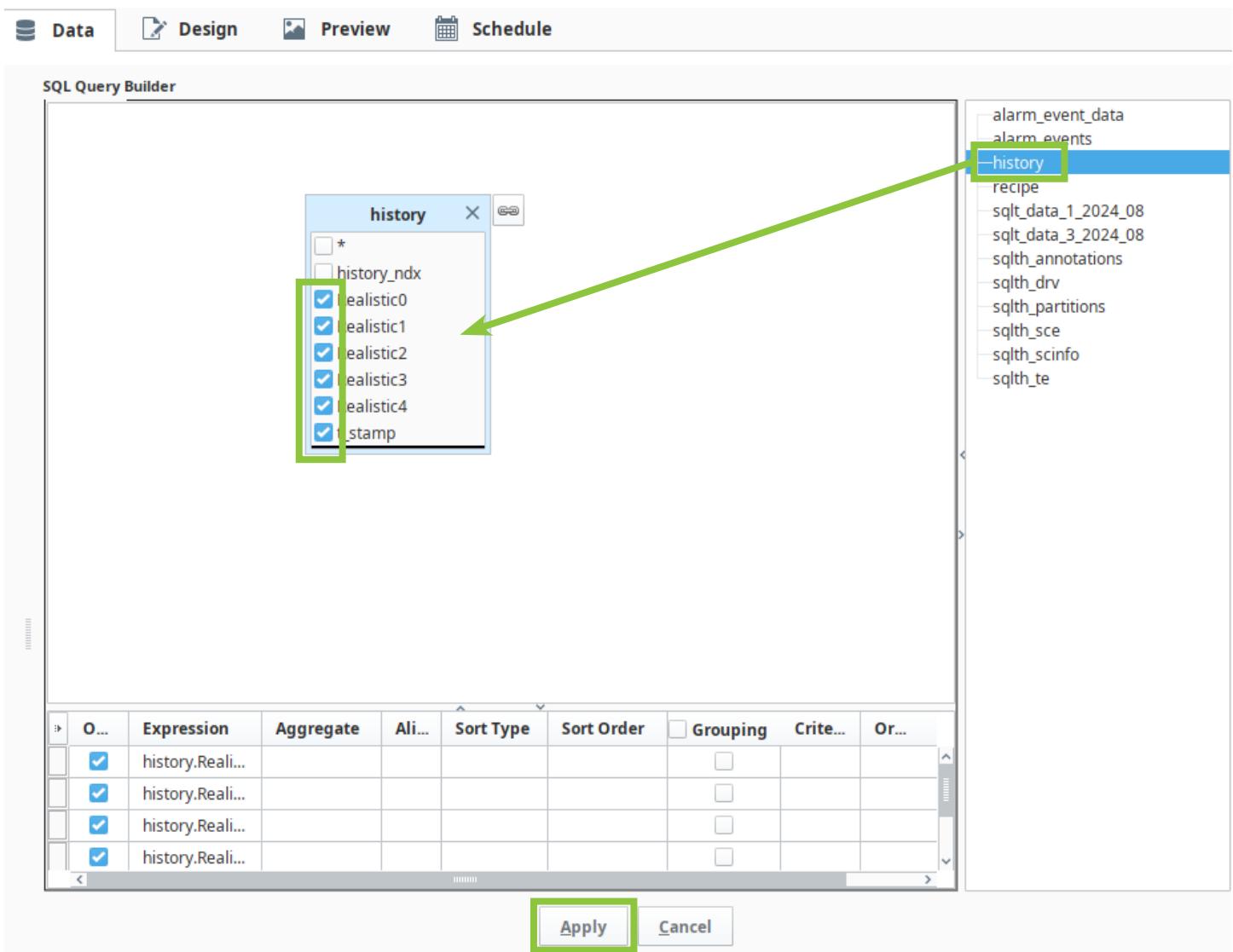


The Query Builder lists all the tables in our database. We can drag tables into the blank space in the query builder, and then interactively add the fields we want for our query and change settings on those fields. We do not need any of those settings for our query, so we won't worry about that section of the query builder.

1. Drag **history** from the table list into the blank area of the Query Builder.
2. Check the box next to **Realistic0**.

Note: Make sure you check the box, and don't just click on the field name.

3. Check the boxes for **Realistic1**, **Realistic2**, **Realistic3**, **Realistic4**, and **t_stamp**.
4. Click **Apply**.



While the Query Builder is useful for adding fields to a query (and is particularly useful for longer, more complex queries that need multiple tables), it unfortunately does not allow you to filter the table results based on the report's parameters, so we will need to write a bit of SQL ourselves.

If you have never written SQL before, don't worry. It's a much easier language to deal with than Python.

Two important things to note about SQL for our purposes: first, the language is case-insensitive, so while we will be putting SQL keywords in all caps, that isn't strictly necessary. And second, SQL is whitespace insensitive. Again, we'll be spacing out the query for readability, but it will work without the extra lines.

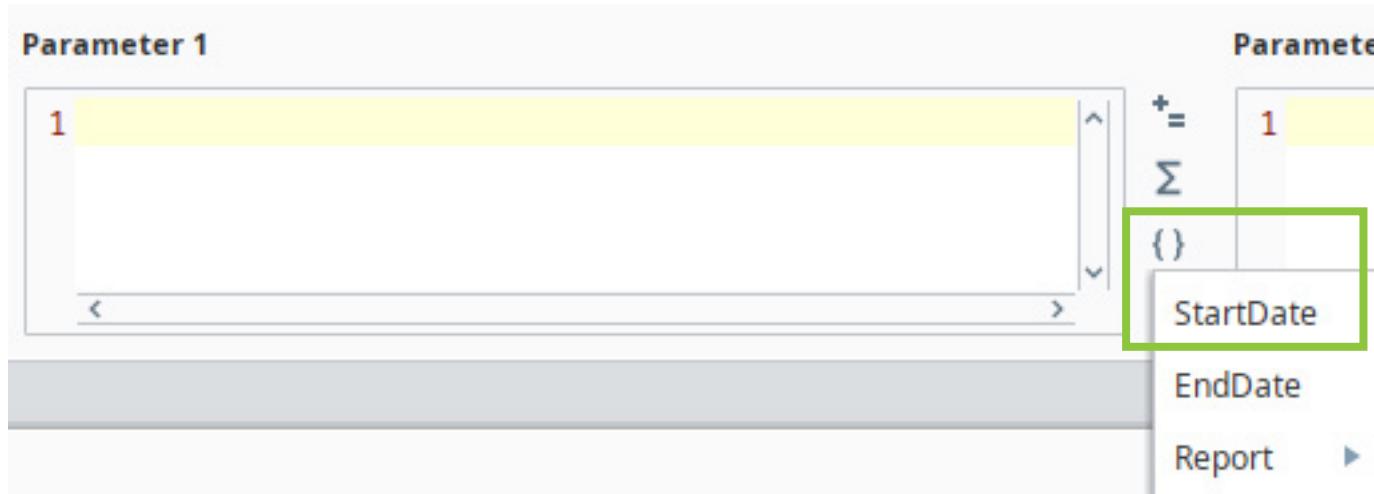
1. Click at the end of the query.
2. Press **Enter**.

3. Enter WHERE t_stamp BETWEEN ? AND ?

The WHERE clause in SQL adds a filter to the query. Because dealing with dates can be challenging, the language includes a BETWEEN/AND clause that returns data from the given field between the two given dates. The question marks are not part of SQL. Instead, they are for Ignition: the SQL Query type uses question marks to represent parameters to be passed into the query. Once you type these, you will see first one, and then two additional boxes appear at the bottom of the screen, where we can enter in the parameters we want the question marks to represent.

4. Click {} in the **Parameter 1** box.

5. Click **StartDate**.



6. Click {} in the **Parameter 2** box.

7. Click **EndDate**.

Finally, we want to limit the data being returned. If your Transaction Group has been running since Tuesday, you likely have many thousands of records by now. While there might be times when it makes sense to generate a report with a lot of data, we generally want to try to limit that, as generating an overly large PDF can cause your computer to hang.

The code we will be adding here is specific to Microsoft SQL Server. While much of SQL is standard, each database engine employs slight variations in the language, and this is one such example. If you are using a database other than SQL Server, you will want to search online for the proper code to limit results.

8. Enter TOP 1000 after SELECT on the first row of the query.

9. Ensure your final query looks like this:

```

SELECT TOP 1000 history.Realistic0,
       history.Realistic1,
       history.Realistic2,
       history.Realistic4,
       history.Realistic3,
       history.t_stamp
FROM history
WHERE t_stamp BETWEEN ? AND ?

```

Note: The order in which the fields are listed after SELECT is unimportant.

10. Click **File**.
11. Click **Save All**.

Test the Query

Now that we have written our query, let's test to make sure it is working correctly.

1. Click the **Preview** tab.

```

1 <?xml version="1.0" ?>
2 <sample-data>
3   <!--This is the raw data used to c
4   <EndDate>2024-08-14 20:57:28.684</
5   <Report>
6     <Gateway>Ignition-TR-2RXX3W3-L
7     <Name>Basic Report</Name>
8     <Path>Basic Report</Path>
9     <Timestamp>2024-08-14 20:57:28
10    </Report>
11    <StartDate>2024-08-14 12:57:28.684
12    <history>
13      <row-0>
14        <Realistic0>102.2206553189
15        <Realistic1>100.0019889460
16        <Realistic2>98.40102595332
17        <Realistic4>88.05043562875

```

You will get a blank page but should see the sidebar populating with data. You will notice an error across the bottom if there is a problem.

Note that the Preview limits the results to 100 records, regardless of how much data is being returned by the query. When we run the final report, we will get the full 1000 records.

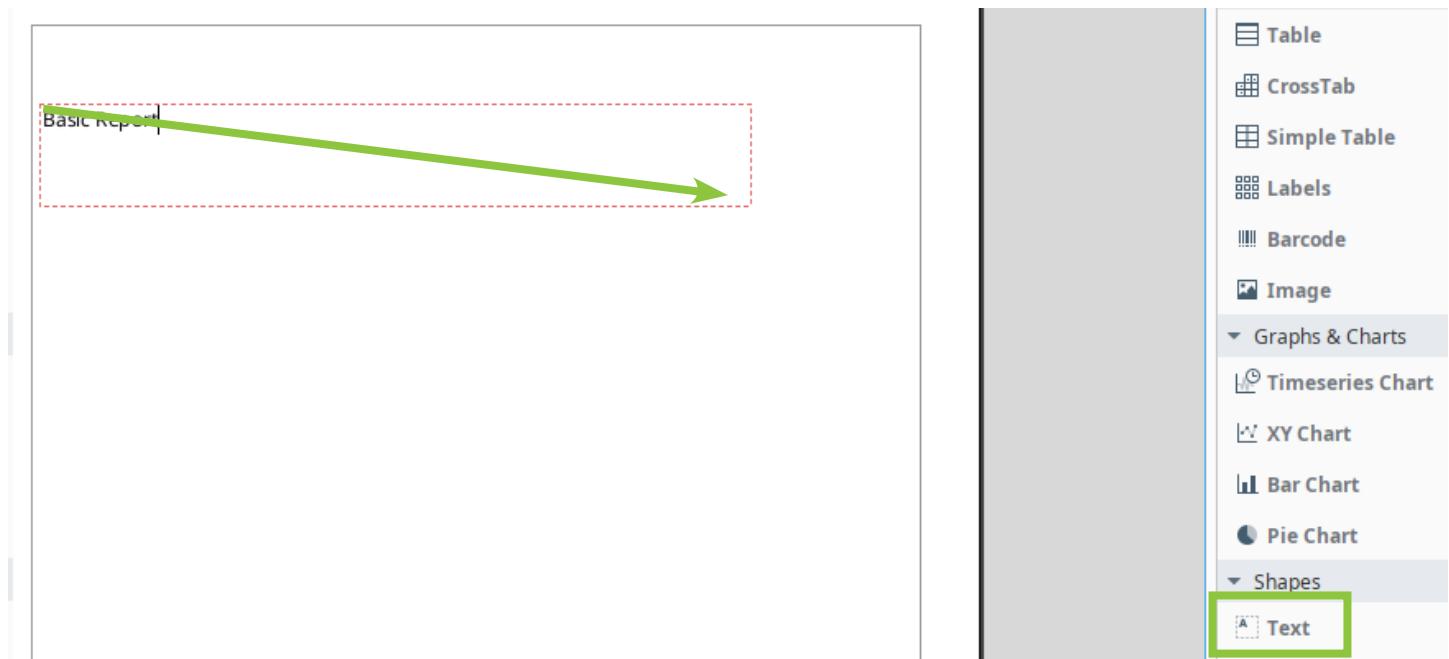
Designing the Report

Now that we have our data source set up and tested, it's time to design the report.

1. Click **Design**.
2. Click **Text**.
3. Click-and-drag in the page to create a **text shape**.

Note: The text tool in reports is a drawing tool, similar to what we saw in Vision's drawing tools (see "Explore the Vision Drawing Tools" on page 110), rather than a component. So, you need to click the tool, then click-and-drag in the design space, rather than dragging the tool to the design space.

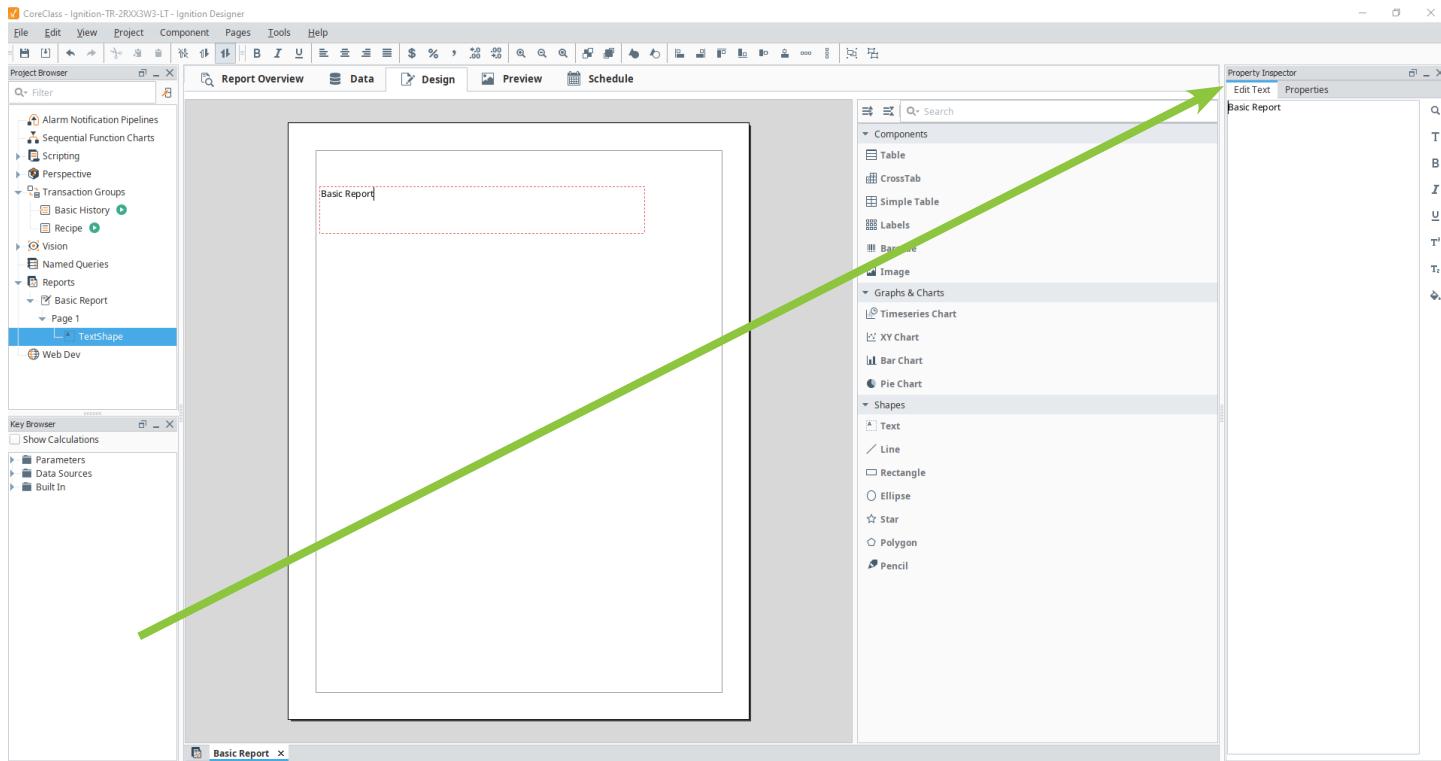
4. Enter **Basic Report**.



Customize the Design Interface

As with other modules, Reports default to having the Property Editor in the bottom left corner of the screen, and just like before, we can modify the layout of the panels to make things a bit more useful.

1. Press and hold your mouse down on the title bar of the **Property Inspector**.
2. Click and drag across the screen to dock the Property Inspector on the right side.



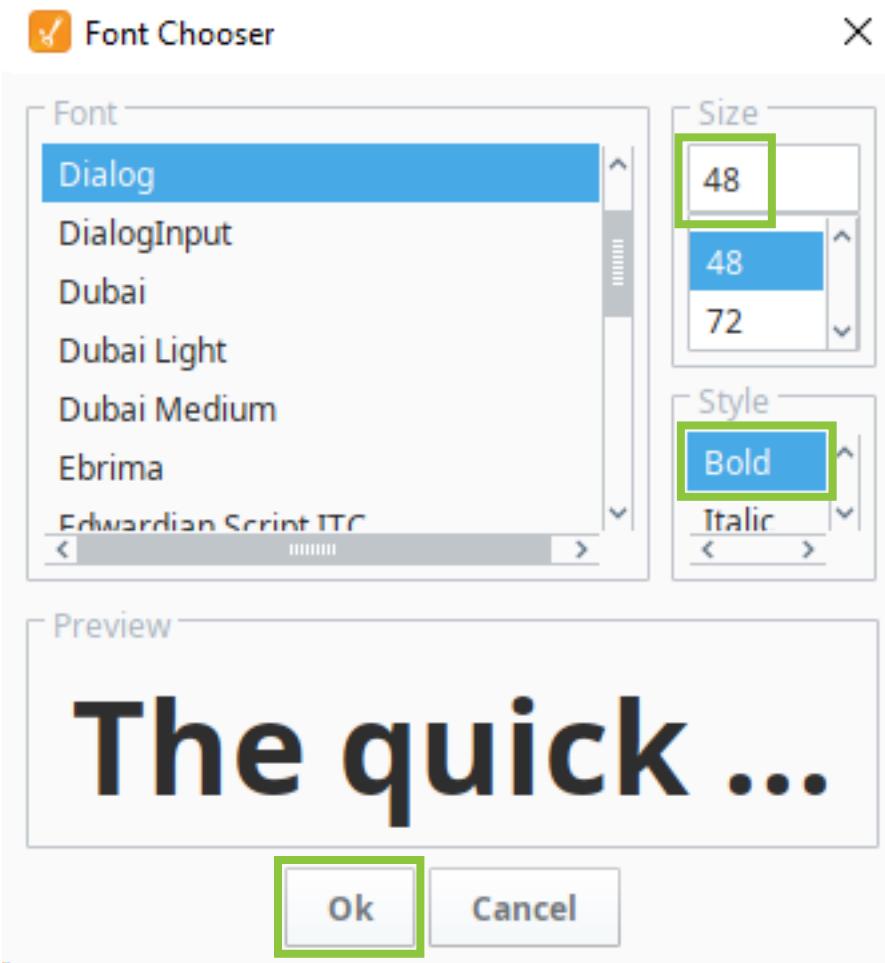
Note: One difference in Reports from other modules is that the Components is not a movable panel, so you must dock the Property Inspector to its right.

Format Text

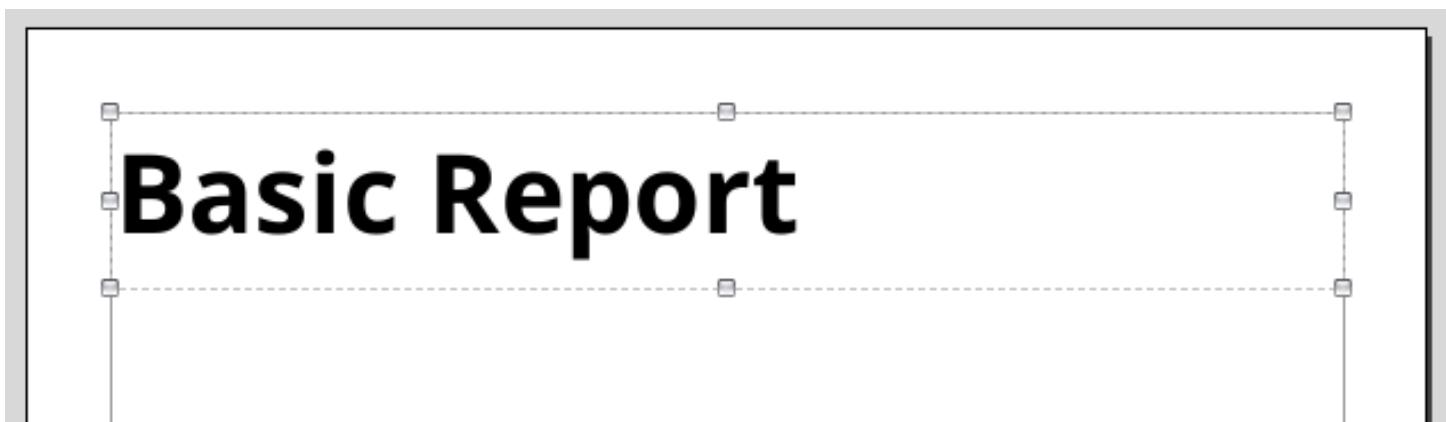
Now that our Property Inspector is in a more usable location, we can use it to format the title.

3. Click the **text box** on the page.
4. Click **T** on the Property Inspector.
5. Click **48** in the Size box.
6. Click **Bold** in the Style box.

7. Click **OK**.



8. Click in a blank area of the page.
9. Click the **text box**.
10. Drag the text box to the top left corner of the margins.
11. Resize the text box to stretch across the page and to be just large enough to fit the text.

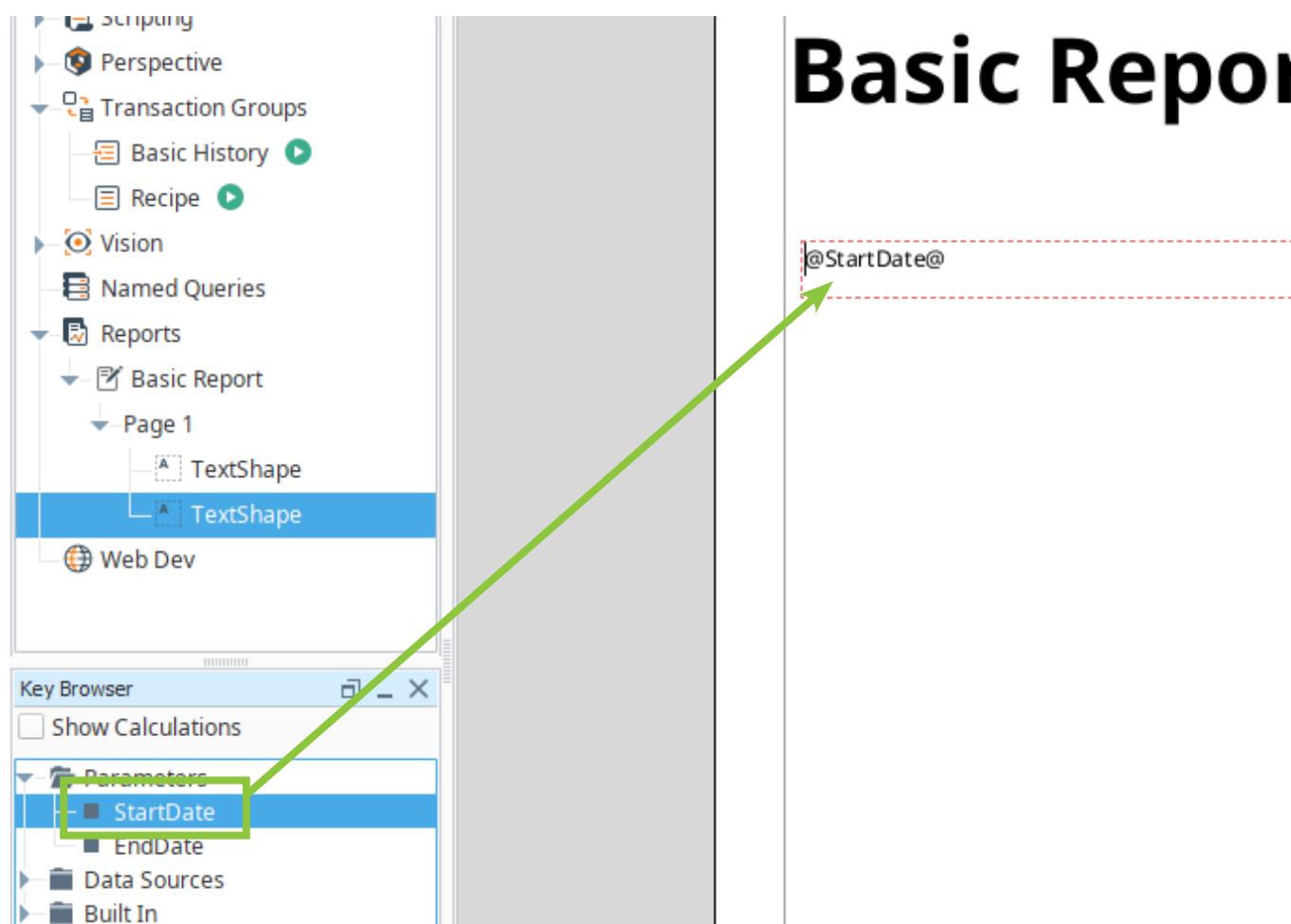


Add Dynamic Text

In addition to static text, our reports can have dynamic text to display things like the date range represented on the report and the page number. Both of these, and many other parameters, are accessible in the Key Browser panel.

When combining dynamic text with static text or other dynamic text, Reports use a special syntax called a Keychain Expression. This represents dynamic references with the @ symbol, and can be typed directly into text shapes or the Edit Text area of the Property Inspector.

12. Click **Text**.
13. Click and drag to draw a **text box** on the page.
14. Expand **Parameters**.
15. Drag **StartDate** to the text box.

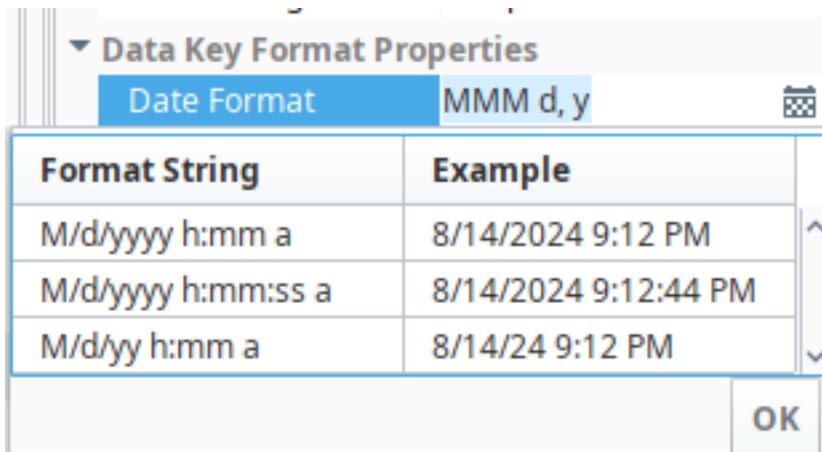


16. Click in the **text box**.
17. Enter a space after **@StartDate@**.

18. Enter **to**.
19. Enter another **space**.
20. Drag **EndDate** to the text box.
21. Ensure your text field looks like this:

@StartDate@ to @EndDate@

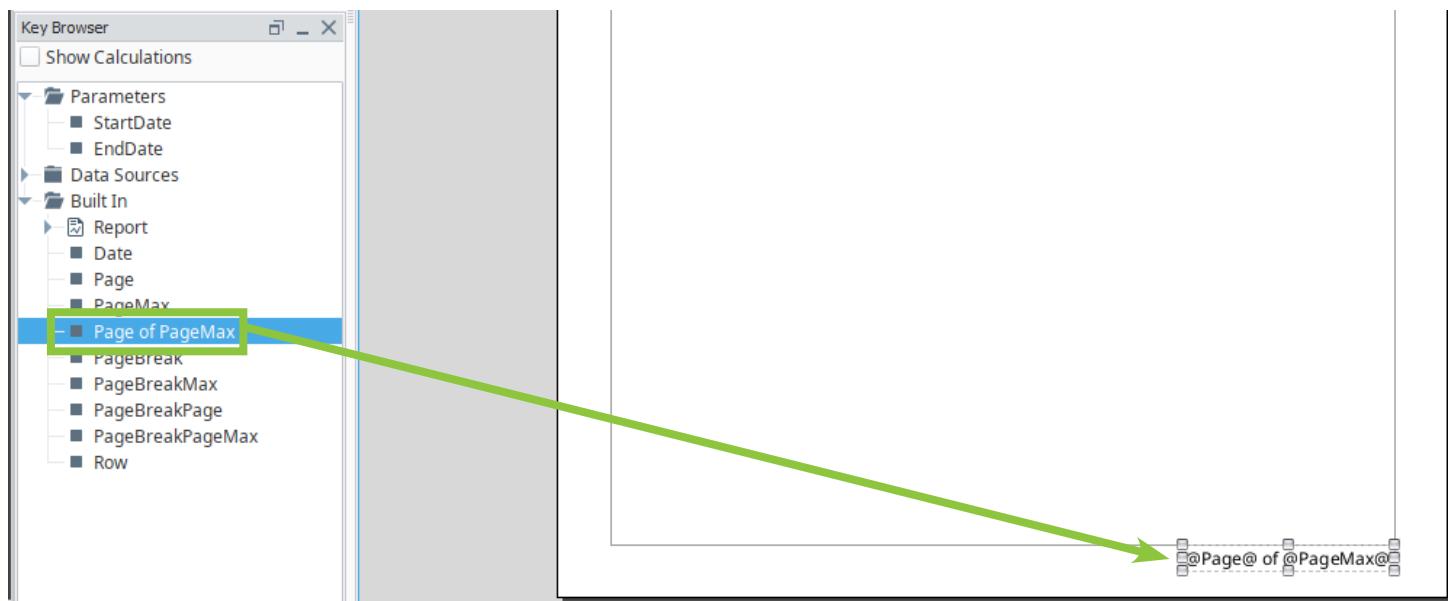
22. Click **Properties**.
23. Click the **calendar icon** on the **Date Format** property.
24. Click a date format that displays both the date and the time.
25. Click **OK**.



26. Click a **blank area** of the report.
27. Click on the **text box**.
28. Drag and resize the text box to stretch the width of the page and rest below the header.



29. Expand **Built In** in the Key Browser.
30. Drag **Page of PageMax** to the bottom right corner of the page, just below the margin.

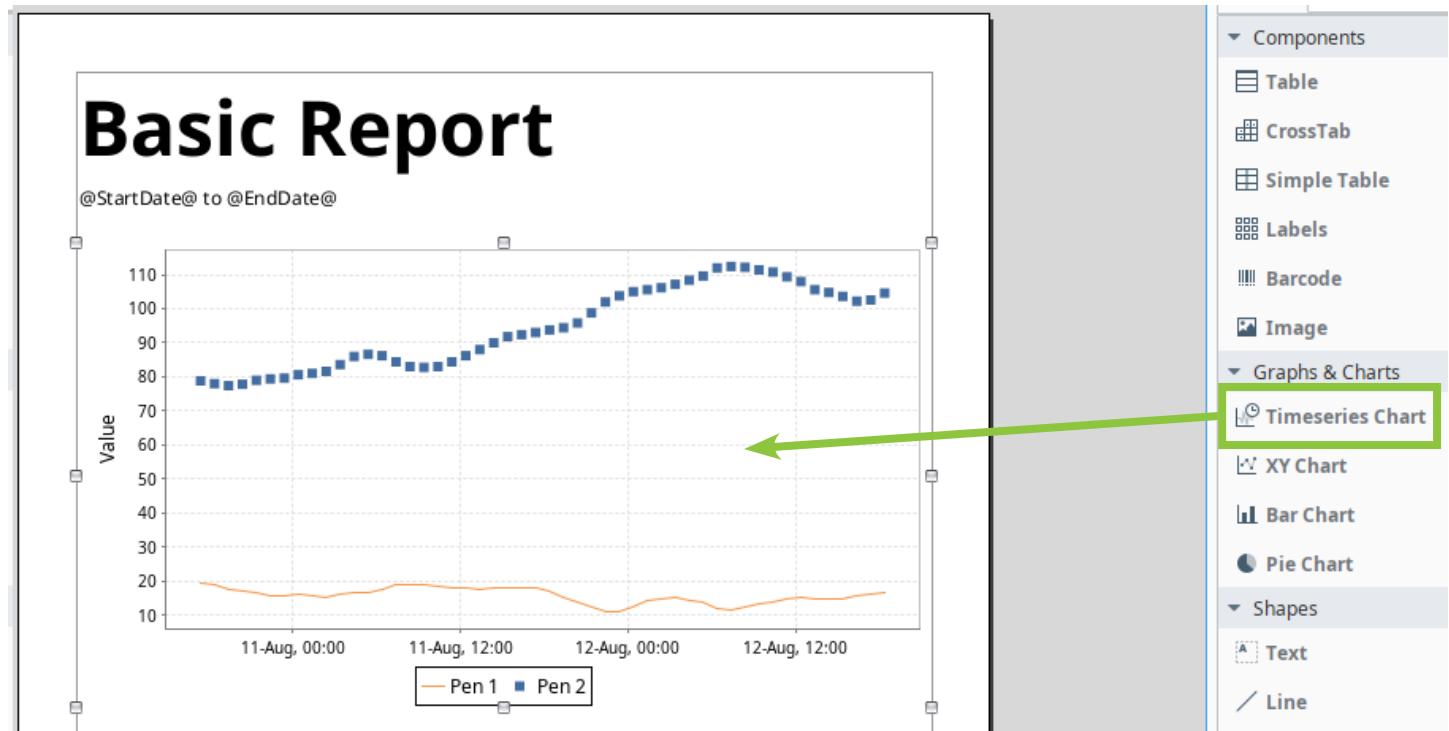


31. Click **Properties**.
32. Select **Right** from the **Horizontal Alignment** list.
33. Click **Preview**.
34. Click **File**.
35. Click **Save All**.

Add a Chart

We will represent the data being returned by the query in two ways. First, we will chart the two of the Realistic Tags. Then, we will add a table.

1. Click **Design**.
2. Drag **Timeseries Chart** to the Window.
3. **Size and position** the chart to fill the top half of the page.

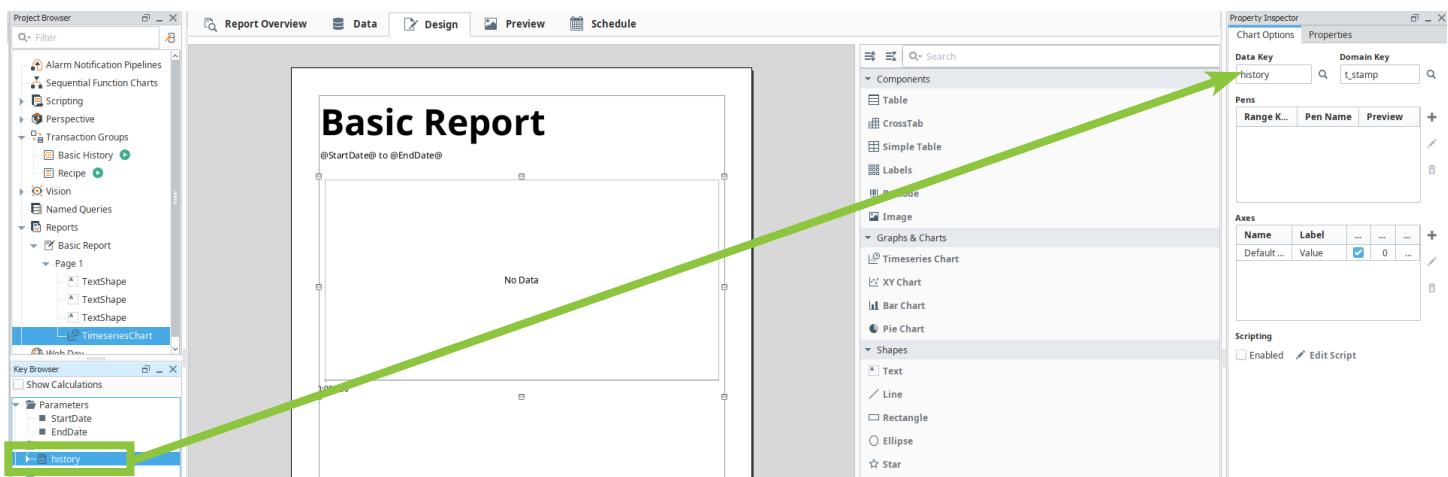


The Timeseries Chart has two pens on it representing sample data. It is best to delete these pens before adding out own.

4. Click **Test1** on the Pens table.
5. Click .
6. Click **Test2** on the Pens table.
7. Click .

The order in which you add data to the Timeseries Chart is very important. You must add the Data key to the chart first, before you add pens. Adding the data key associates a particular data set with the chart, which it can then use to draw the pens. If you add the pens first, they will not properly associate with the data key, and will not draw on the chart. Once you have the data key set, you can drag fields from the Key Browser to the Pens table.

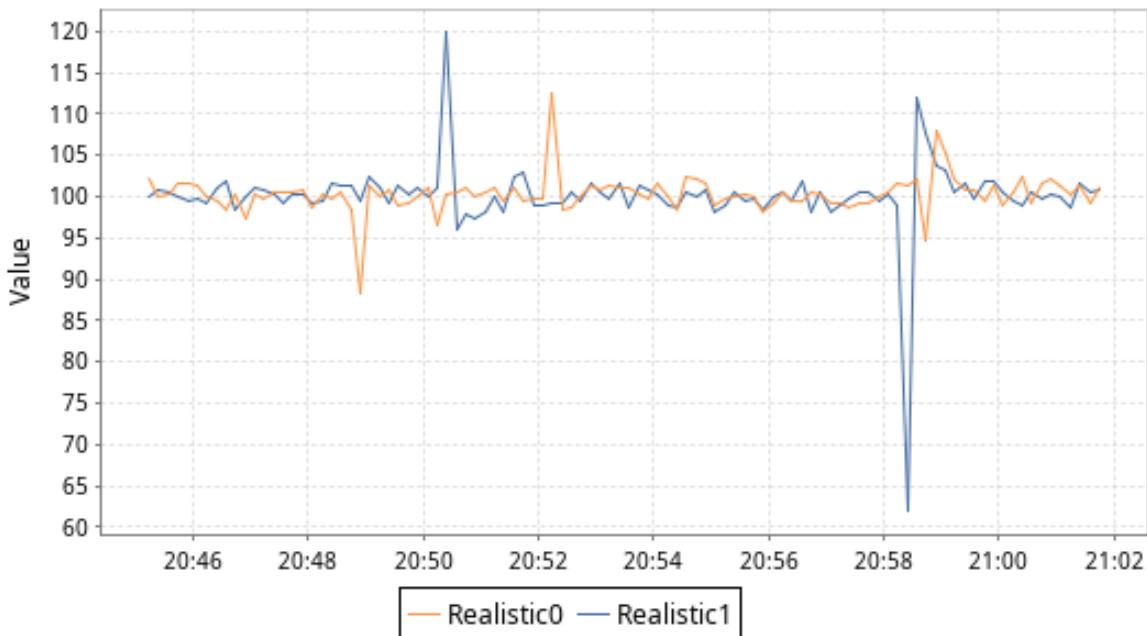
8. Expand **Data Sources** in the Key Browser.
9. Drag **History** to the **Data key field** in the Chart Options.



10. Expand **history** in the Key Browser.
11. Drag **Realistic0** to the Pens table.
12. Drag **Realistic1** to the Pens table.
13. Click **Preview**.

Basic Report

Aug 14, 2024 to Aug 14, 2024



Add a Table

Now that we have some of our data represented in a chart, we want to add all the data to a table.

Tables in a report will automatically repeat across as many pages as needed to display all the rows. In our case, this should result in a report that is about 70 pages long, although if you had to restart your Transaction Group at the beginning of this lesson, your report may be shorter.

1. Click **Design**.
2. Drag **Table** to the page.
3. **Position and size** the table to fill the bottom half of the page.

The screenshot shows a report design interface with a main canvas and a sidebar of components.

Main Canvas:

- Title:** Basic Report
- Text:** @StartDate@ to @EndDate@
- Chart:** A line chart titled "Realistic0" and "Realistic1" showing data from August 11 to 12. The Y-axis is labeled "Value" and ranges from 10 to 60. The X-axis shows dates: 11-Aug, 00:00, 11-Aug, 12:00, 12-Aug, 00:00, 12-Aug, 12:00. The blue line (Realistic1) starts at ~40, dips to ~38, rises to ~42, dips to ~38, rises to ~55, dips to ~52. The orange line (Realistic0) starts at ~23, dips to ~10, rises to ~15, dips to ~10, rises to ~25, dips to ~22, rises to ~30, dips to ~32, rises to ~42.
- Text:** Objects Details
- Page Number:** @Page@ of @PageMax@

Sidebar Components:

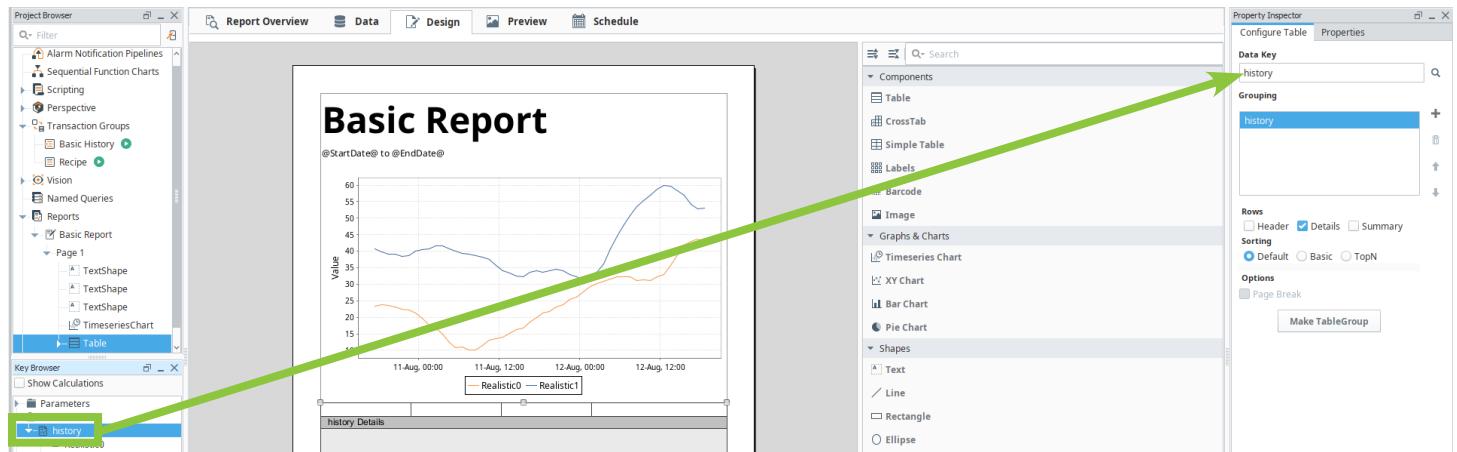
- Components:** Table (highlighted with a green border)
- Graphs & Charts:** Timeseries Chart, XY Chart, Bar Chart, Pie Chart
- Shapes:** Text, Line, Rectangle, Ellipse, Star, Polygon, Pencil

A large green arrow points from the sidebar's "Table" icon towards the empty area below the chart on the main canvas.

Configure the Table

As with charts, it's important to make sure to set the Data key for the table before adding any data to it.

1. Drag **history** from the Key Browser to the **Data key field** on Configure Table.

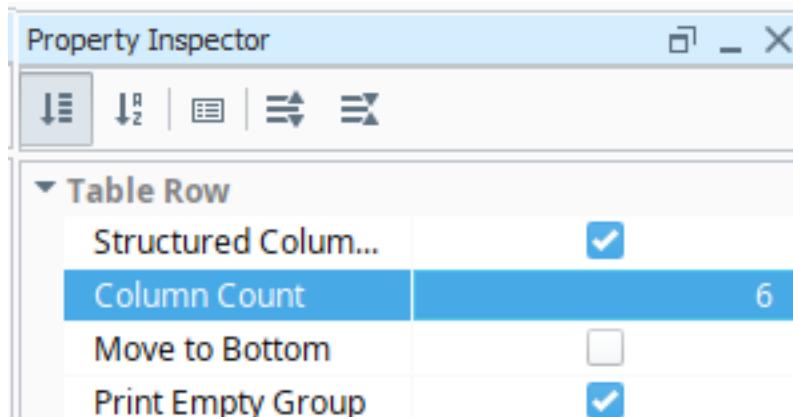


Before we add our data, we want to note that the table initially only has four columns. Our data, however, has six fields: the five Realistics and t_stamp. So, our next step is to add columns to the table. To do this, we first need to select the Details row.

2. Double-click **history Details** on the table.

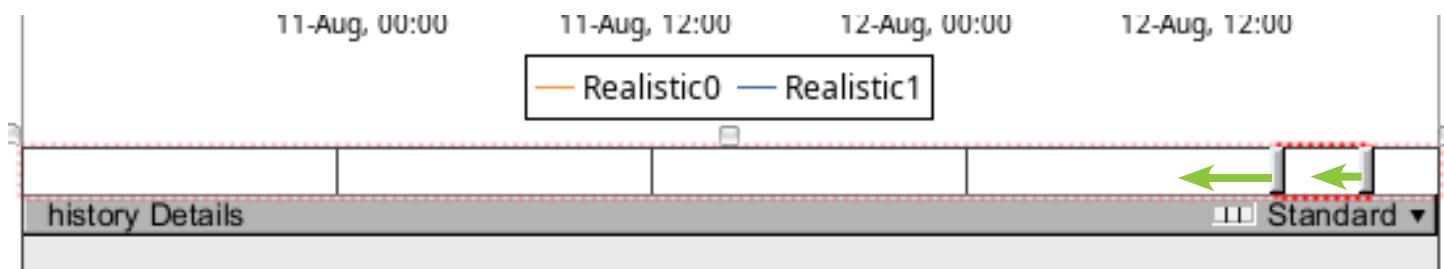
At the top of the Property Inspector, you will see a Column Count property. We want to change this to 6. However, you will not immediately see a difference on the table itself.

3. Enter **6** in the **Column Count** field.
4. Press **Enter**.



To see the new cells, we need to drag to resize them in the table.

5. Click in the **right-most** cell of the row.
6. Drag the **small gray bar** at the right side of the cell.
7. Repeat this to **resize** the sixth column.

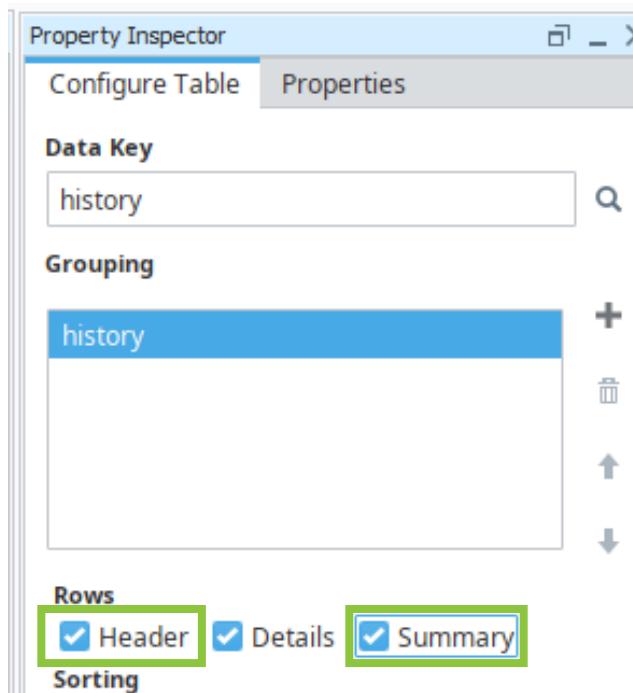


Add Header and Summary Rows

Deciding how big the columns need to be will require that we look at the actual data to be displayed, but there are a few more things we want to do first.

We never want unlabeled numbers. To make the table clearer, we want to add a header row. This row will automatically repeat across every page. We also want to add a summary row, which will display as the final row of the table on the last page of the report.

1. Click the blank gray area of the table.
2. Check **Header** in the Rows section.
3. Check **Summary**.

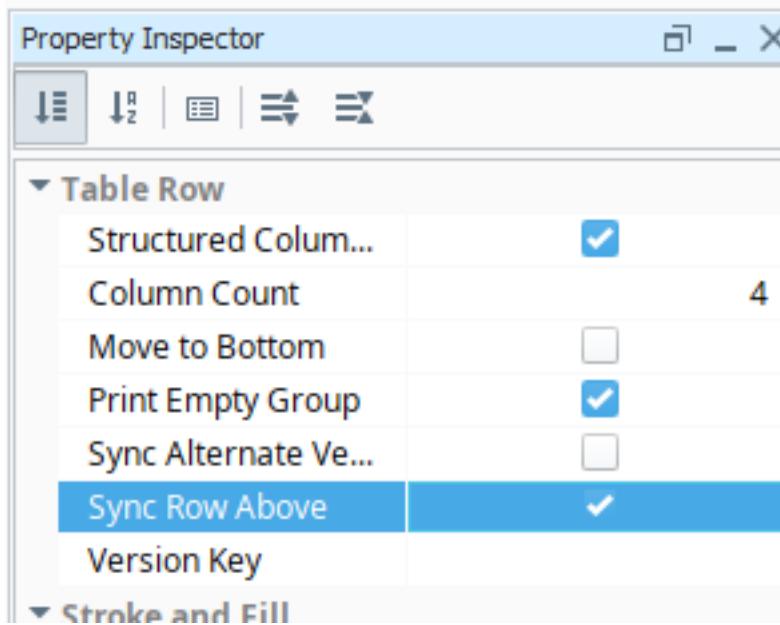


As with the details row, the header and summary rows default to 4 columns. We need to manually set both of them to 6 columns.

4. Double-click **history Header** on the table.
5. Enter **6** in the **Column Count** field.
6. Press **Enter**.
7. Double-click **history Summary** on the table.
8. Enter **6** in the **Column Count** field.
9. Press Enter.
10. Click in the **right-most** cell of the **Summary** row.
11. Drag the **small gray bar** at the right side of the cell.
12. Repeat this to **resize** the sixth column.

You will notice as you resize a column in one row that it does not resize the same column in the other rows. While there are times in the advanced reports that while is useful, for our report we want all the columns to be the same size. We can do that by synchronizing each row with the row above it.

13. Double-click **history Summary**.
14. Click **Sync Row Above** in the Property Inspector.



15. Double-click **history Details**.

16. Click **Sync Row Above** in the Property Inspector.
17. Resize any column in the **Summary row**.

For now, having that setting enabled is enough. We will size the columns more precisely once we have data in them.

Add Data

Adding data to the table is pretty easy—we just need to drag the fields from the Key Browser to the cell on the table's Details row where we want that data.

1. Drag **Realistic0** to the first cell of the Details row.

history Header	Realistic0@	history Details	history Summary

2. Repeat to drag the other **Realistics** to the additional cells.
3. Drag **t_stamp** to the right-most cell.

4. Enter **Realistic 0** in the first cell of the Header row.
5. Repeat to **add headers** to the other cells.
6. Click **Preview**.



Realistic 0	Realistic 1	Realistic 2	Realistic 3	Realis tic 4	Time stamp
102.22	100	98.4	98.91	88.05	Aug 14, 2024
100.03	100.67	101.4	98.45	92.32	Aug 14, 2024
100.32	100.57	98.86	100.45	94.46	Aug 14, 2024
101.71	100.07	101.81	100.9	95.76	Aug 14, 2024
101.66	99.36	99.33	102.24	99.16	Aug 14, 2024
101.34	99.82	99.71	102.42	98.09	Aug 14, 2024
100.1	99.31	98.17	102.92	185.5 4	Aug 14, 2024

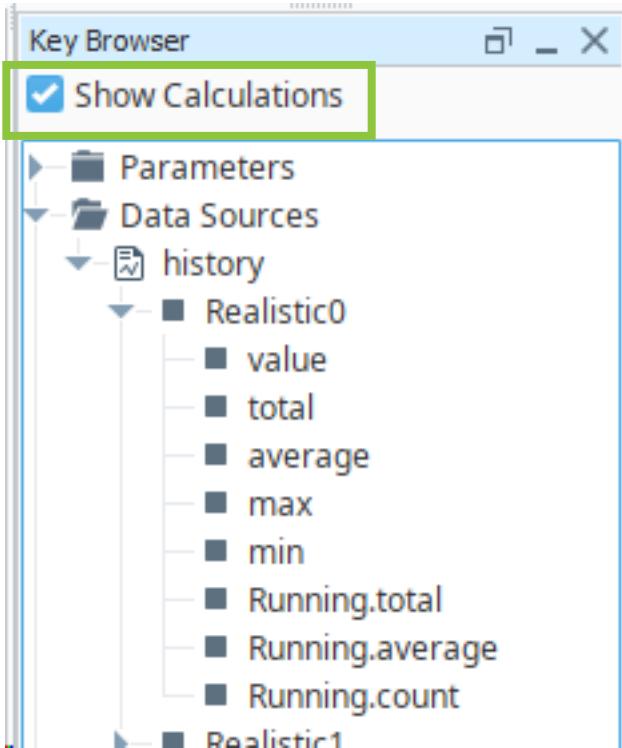
1 of 15

You will now see the data in the table. You can use the buttons at the bottom of the screen to preview additional pages.

Add Summary Data

Let's add summary information.

1. Click **Design**.
2. Click **Show Calculations** at the top of the Key Browser.
3. Expand **Data Sources**.
4. Expand **history**.
5. Expand **Realistic0**.



You will see that enabling the Show Calculations setting gives use summary information about each field.

6. Drag **total** to the first cell in the Summary row.
7. Expand **Realistic1**.
8. Drag **average** to the second cell in the Summary row.
9. Repeat to add **max** to the summary row for **Realistic2**, and **min** for **Realistic3**. We will not add a summary for Realistic4.
10. Click **Preview**.
11. Click the  button.

You will see the summary information for each column at the bottom of the table.

Format the Table

One final step: we want to do a bit of formatting to make the data look better.

1. Click **Design**.
2. Click the first cell of the **Header** row.
3. Press and hold **Shift**.

4. Click the last cell of the **Header** row.
5. Click **B** on the Property Inspector.
6. Click **t_stamp** on the Details row.
7. Click **Properties**.
8. Click the **calendar icon** on the Date Format property.
9. Select a **date format** that shows both date and time.
10. Click **OK**.
11. Use the **gray bars** in the **Summary row** to resize the columns. The columns for the Realistics can be relatively narrow, giving the t_stamp column room to display the much longer date.
12. Click **Preview**.

Realistic 0	Realistic 1	Realistic 2	Realistic 3	Realistic 4	Time stamp
102.22	100	98.4	98.91	88.05	8/14/2024 8:45:14 PM
100.03	100.67	101.4	98.45	92.32	8/14/2024 8:45:24 PM
100.32	100.57	98.86	100.45	94.46	8/14/2024 8:45:34 PM
101.71	100.07	101.81	100.9	95.76	8/14/2024 8:45:44 PM
101.66	99.36	99.33	102.24	99.16	8/14/2024 8:45:54 PM
101.34	99.82	99.71	102.42	98.09	8/14/2024 8:46:04 PM
100.1	99.31	98.17	102.92	185.54	8/14/2024 8:46:14 PM
99.41	101.05	101.05	137.1	142.58	8/14/2024 8:46:24 PM
98.41	101.73	100.59	83.91	91.37	8/14/2024 8:46:34 PM
100.11	98.31	101.44	80.38	67.04	8/14/2024 8:46:44 PM
97.31	99.96	99.36	88.75	79.78	8/14/2024 8:46:54 PM
100.12	100.95	100.03	93.25	88.15	8/14/2024 8:47:04 PM
99.61	100.73	98.23	95.46	94.03	8/14/2024 8:47:14 PM
100.42	100.23	99.3	98.09	96.23	8/14/2024 8:47:24 PM
100.61	99.08	100.2	100.04	97.1	8/14/2024 8:47:34 PM
100.55	100.34	99.12	143.47	98.36	8/14/2024 8:47:44 PM

1 of 7

13. Click **File**.
14. Click **Save All**.

Report Scheduling

You can add as many schedules as you want to your report. Schedules allow you to make your report automatically save, email, and more at times you specify.

Create a Schedule

We want our report to save to a file every night at 1 am.

1. Click **Schedule**.
2. Click **+** in the upper right corner.

Below our table are three tabs: Schedule, Parameters, and Actions. We need to work our way through them to complete the schedule.

The Schedule tab allows you to set how often the report is run using a CRONTAB scheduling system. CRONTAB is a scheduling system used by system administrators to specify when jobs should run, as an offset of midnight. In our case, we only need to change the Hours setting to offset by one hour, or 1 am.

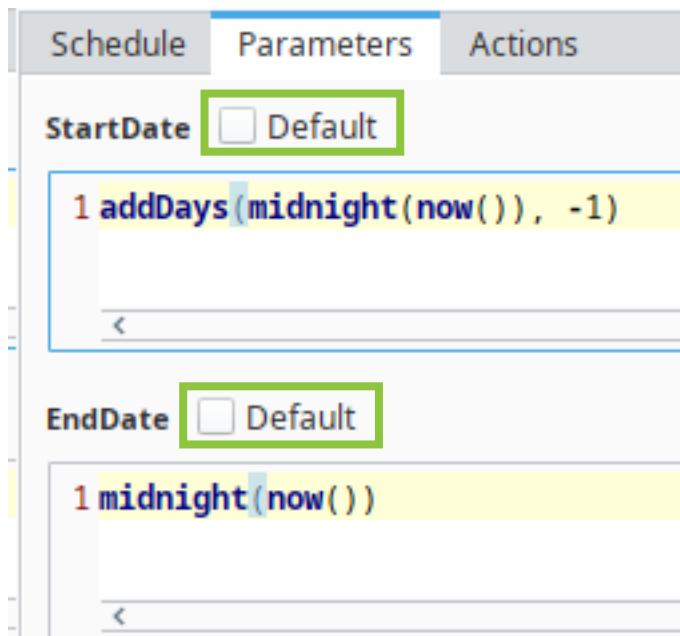
3. Enter 1 in the Hours field.

The screenshot shows the 'Schedule' tab selected in a navigation bar with three tabs: Schedule, Parameters, and Actions. Below the tabs is a section titled 'Schedule' with a sub-section 'Common Settings'. Under 'Common Settings', there is a dropdown menu set to 'Custom'. The 'Minutes' section shows a value of '0' in a dropdown menu next to a checked checkbox. The 'Hours' section shows a value of '1' in a dropdown menu next to a checked checkbox. A green box highlights the '1' in the 'Hours' dropdown. The 'Days' section shows a value of '*' in a dropdown menu next to a checked checkbox. The 'Months' section shows a value of '*' in a dropdown menu next to a checked checkbox. At the bottom, there are three dots (...).

Set Parameters

The Parameters tab allows you to change the defaults for the data source's parameters. In this case, we want a daily report to show the entire day, not just the last 8 hours, so we need to enter a different expression for our start and end dates.

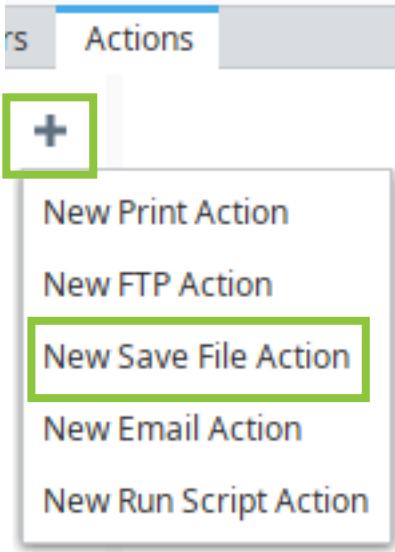
1. Click **Parameters**.
2. **Uncheck Default** for StartDate.
3. Enter `addDays(midnight(now()), -1)`
4. **Uncheck Default** for EndDate.
5. Enter `midnight(now())`



Set Report Actions

The Actions tab allows us to set a number of actions to happen each time this schedule is run. In our case, we want to save this report to the hard drive.

1. Click **Actions**.
2. Click **+**.
3. Click **New Save File Action**.

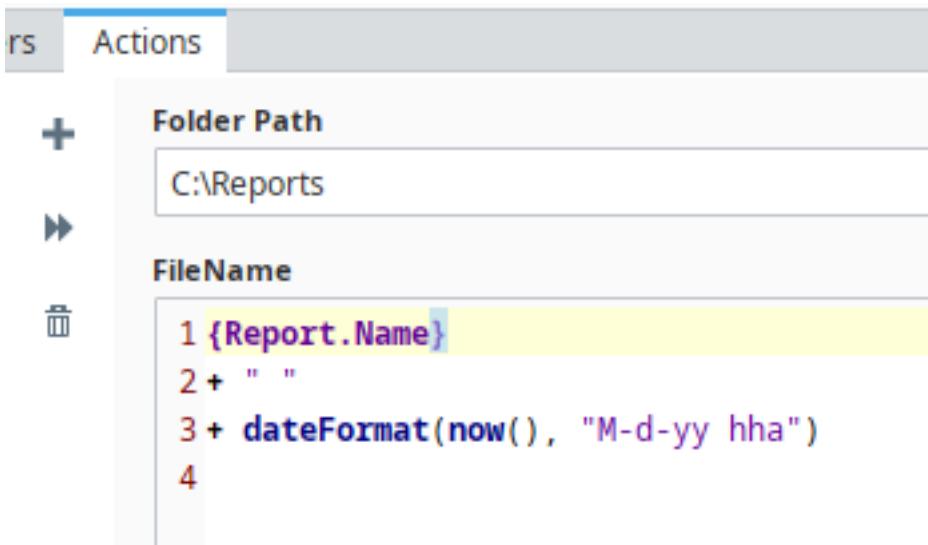


4. Enter a **valid file path** in the Folder Path field.

The default expression for the File Name takes the name of the report and appends a time stamp, ensuring that each report will have a unique name. For our purposes, this is fine.

Rather than wait until 1am to run this report, we can force an immediate execution, and then go to Windows and view the report.

5. Click ➤ .



6. Open **Windows File Explorer**.

Note: You can use your keyboard to do this by pressing and holding the Windows key and then pressing E.

Navigate to the folder you set in the Folder Path above.

7. Double-click the **report**.

Note: You will need Adobe Acrobat or some other tool to read the PDF.

Client Reports

You can allow your users to dynamically view reports on any Window in your Client.

1. Click **Vision**.
2. Expand **Windows**.
3. Expand **Main Windows**.
4. Right-click **Empty**.
5. Click **Duplicate**.
6. Right-click **Empty (1)**.
7. Click **Rename**.
8. Enter **Report**.
9. Press **Enter**.
10. Expand the **Components Palette**.
11. Enter **report** in the Filter box.
12. Drag a **Report Viewer** component to the Window.
13. Right-click the **Report Viewer**.
14. Click **Layout**.
15. Click **Anchored**.
16. Click the **top**, **right**, and **left** anchors.
17. Click **OK**.
18. Select **Basic Report** from the Report Path list.

19. Size and position the **Report Viewer** to display the width of the report. Leave about an inch of space above it.

Add a Date Range

We want our operators to be able to set the length of time they want to display in the report. We can do this by adding a Date Range component, and then binding it to the Report's parameters.

1. Expand the **Components Palette**.
2. Enter **date** in the Filter box.
3. Drag a **Date Range** component to the Window.
4. Size and position it above the Report Viewer.
5. Right-click the **Date Range**.
6. Click **Layout**.
7. Click **Anchored**.
8. Click the **top**, **right**, and **left** anchors.
9. Click **OK**.
10. Click the **Report Viewer**.
11. Click  for the Report's **Start Date** parameter.
12. Click **Property**.
13. Expand **Date Range**.
14. Click **Start Date**.
15. Click **OK**.
16. Click  for the Report's **End Date** Parameter.
17. Click **Property**.
18. Expand **Date Range**.
19. Click **End Date**.
20. Click **OK**.

 Report


8/28/24 - 8/30/24



Jul 31

Aug 5

Aug 10

Aug 15

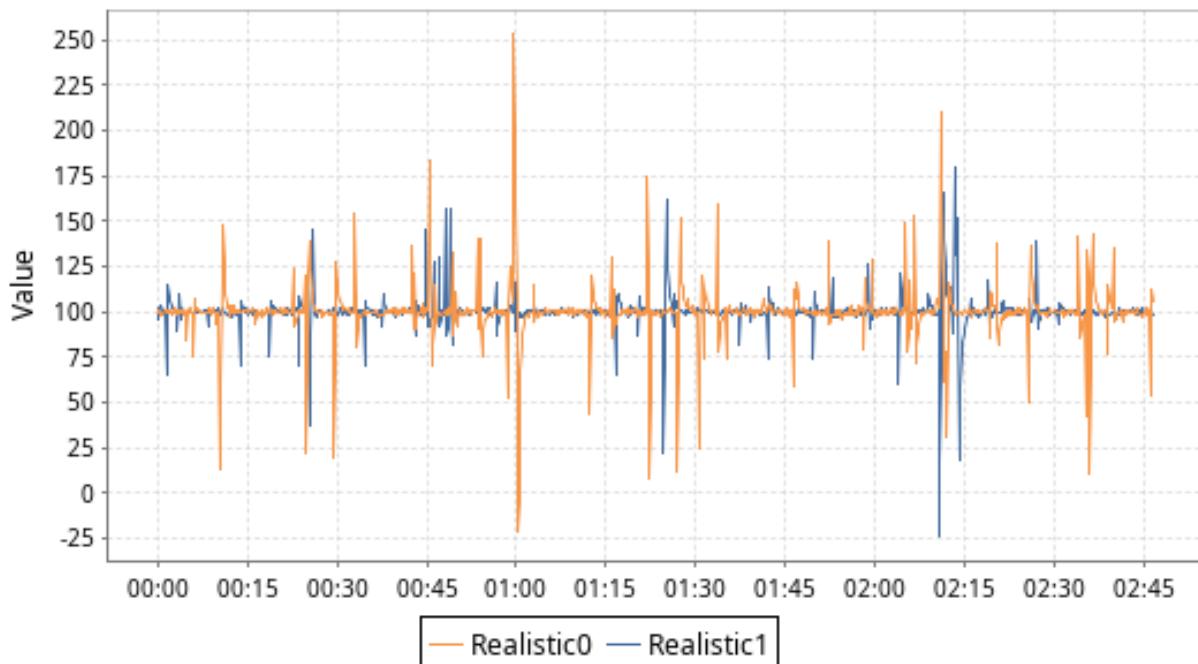
Aug 20

Aug 25

Aug 30

Basic Report

Aug 28, 2024 to Aug 30, 2024


Realistic 0 Realistic 1 Realistic 2 Realistic 3 Realistic 4 Time stamp

100.09	101.6	96.33	100.25	101.01	8/28/2024 12:00:04 AM
100.25	95.52	98.43	99.05	100	8/28/2024 12:00:14 AM
97.79	103.32	100.04	101.51	99.69	8/28/2024 12:00:24 AM
99.45	102.91	100.02	101.51	102.16	8/28/2024 12:00:34 AM
98.79	100.53	100.63	99.08	99.52	8/28/2024 12:00:44 AM
98.38	99.36	99.61	99.36	100.01	8/28/2024 12:00:54 AM
100.53	100.09	98.71	100.13	100.14	8/28/2024 12:01:04 AM
100.58	100.19	101.16	99.19	100.28	8/28/2024 12:01:14 AM
100.47	65.43	99.06	26.68	99.04	8/28/2024 12:01:24 AM
99.73	112.88	45.1	74.82	101.53	8/28/2024 12:01:34 AM
99.88	114.92	96.1	125.44	99.83	8/28/2024 12:01:44 AM



Add the Report Viewer to Navigation

1. Double-click **Navigation** in the Project Browser.
2. Double-click the **Tab Strip**.
3. Click an existing tab.
4. Click **Add Tab**.
5. Select **Main Windows/Report** from the Window Name list.
6. Enter **Report** in the Display Name field.
7. Click **OK**.
8. Click **File**.
9. Click **Save All**.
10. Click **Tools**.
11. Click **Launch Project**.
12. Click **Launch Project (Windowed)**.
13. **Log into** the Client.
14. Click **Report**.
15. Change the date range to see the data update.

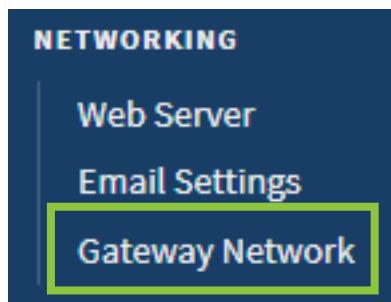
Gateway Network

The Gateway Network allows us to connect Gateways to each other so that they can share data such as Tags, alarms, and history. Making a Gateway Network Connection is easy.

Connect to a Gateway

Let's set up a connection to the instructor's Gateway.

1. Open the **Gateway** web page.
2. **Log in.**
3. Click **Config.**
4. Click **Gateway Network** in the Networking section.



One Gateway will make an outgoing connection, while the other Gateway will accept the incoming connection. In class, you will be making the outgoing connection to the instructor's Gateway while they will accept the incoming connection.

Disable SSL

Your instructor will perform the following steps on their Gateway:

1. Click **General**.
2. Click **Require SSL** to set it to **false**.
3. Click **Save Changes**.

A screenshot of the 'General Settings' page. At the top, there is a horizontal navigation bar with tabs: 'General Settings' (selected), 'Outgoing Connections', 'Incoming Connections', 'Diagnostics', 'Queue Management', and 'Proxy Rules'. Below this, there is a section titled 'Main' with two settings. The first setting is 'Enabled', which has a checked checkbox and the note '(default: true)'. The second setting is 'Require SSL', which has an unchecked checkbox and the note 'If true, only connections that use SSL to encrypt traffic will be allowed. This setting only applies to incoming connections.' followed by '(default: true)'.

Set Up the Incoming Gateway

On your Gateway, perform the following steps:

1. Click **Outgoing Connections**.
2. Click **Create new Outgoing Gateway Connection...**
3. Enter the **Instructor's Gateway IP** address in the Host field.
4. Enter **8088** in the Port field.
5. Click **Use SSL** to connect to the remote machine to set it to false.

Main

Host	<input type="text" value="10.10.111.52"/> The address of the remote server, not including the port.
Port	<input type="text" value="8088"/> The port of the remote server. (default: 8,060)
Enabled	<input checked="" type="checkbox"/> (default: true)
Use SSL	<input type="checkbox"/> Use SSL to connect to the remote machine. (default: true)

6. Scroll down.
7. Click **Create New Outgoing Gateway Connection**.

Accept Connections

The Instructor now needs to accept the connections. Watch as they perform these steps on their Gateway:

1. Click **Incoming Connections**.

2. Click **More**.
3. Click **approve**.
4. Click **Confirm**.

Set Up a Remote Tag Provider

Now that the Gateways are connected through the Gateway Network, you can share data by setting up a Remote Provider.

A Remote Tag Provider allows you to pull a Tag Provider from one Gateway into another, if both are connected through the Gateway Network.

1. Click **Realtime** in the Tags section.
2. Click **Create new Realtime Tag Provider**.
3. Click **Remote Tag Provider (Gateway Network)**.
4. Click **Next >**.

Standard Tag Provider

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

Remote Tag Provider (Gateway Network)

Creates a link to a tag provider on a different system through the Gateway Network.

Next >

5. Click your **instructor's Gateway**.
6. Click **Next**.
7. Click **default**.
8. Click **Next**.
9. Scroll down.
10. Click **Create New Realtime Tag Provider**.

11. Return to **Designer**.
12. Click **Refresh** on the Tag Browser.
13. Select the **instructor's Gateway** from the dropdown list.

Backups

There are two types of backups in Ignition. Gateway backups consist of everything in Ignition except the activation status, and includes all projects, Tags, connection configurations, alarm notification configurations, and more. Project backups are far less inclusive, and only contain project-scoped resources, such as templates, windows, views, transaction groups, or reports.

Gateway Backup

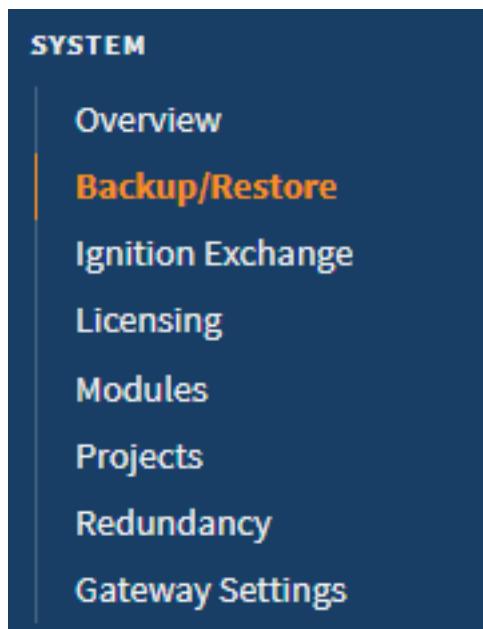
A Gateway backup is an effective way to capture all of the information you need in order to restore a backup in the case of hardware failure, or to transfer an Ignition installation from one server to another.

Generally, the only items not included in a Gateway Backup are licenses, modules, and items stored externally to Ignition, such as database tables and PLC programs. Restoring from a Gateway backup is considered a replacement operation, in that the backup copy will overwrite all resources on the target Gateway.

Download a Backup

Let's download a backup of our Gateway so that you can take all of the work you've done with you after class.

1. Open the **Gateway** web page.
2. Click **Config**.
3. **Log in**.
4. Click **Backup/Restore** in the **System** section.



5. Click **Download Backup**.

Depending on how many resources are in the Gateway, it might take a few moments to download.

The backup will be downloaded to your browser's default download location, which on Windows is usually the Downloads folder.

The filename for a Gateway backup will be the name of the Gateway, combined with "Ignition-backup" and a timestamp. Gateway backups have a .gwbk file extension.

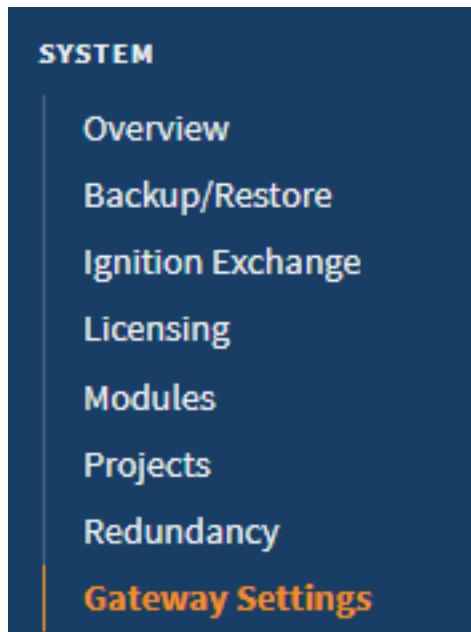
Restore a Gateway

Restoring from a backup is done from the same section of the Gateway web page. Simply select the GWBK file. Note that restoring a backup will always completely overwrite the existing Gateway, so it is a good idea to always download a new backup immediately before restoring.

Schedule a Backup

Any time you are developing in Ignition, it is a good idea to take regular backups. Ignition can even do this for you automatically with the Scheduled Backup Settings.

1. Click **Gateway Settings** in the System section.



2. Scroll down to the **Scheduled Backup Settings**.
3. Check **Enable Scheduled Backups**.
4. Specify a folder to save backups to.

Scheduled Backup Settings	
Enable Scheduled Backups	<input checked="" type="checkbox"/> Enables the scheduled backup system which will automatically make backups at a scheduled time. (default: false)
Backup Folder	<input type="text"/> A path to a folder in which to put the scheduled backups.
Backup Schedule	<input type="text"/> 15 1 * * * A UNIX crontab style scheduling string representing when to make the backups. (default: 15 1 * * *)
Retention Count	<input type="text"/> 5 The number of backups to keep in the backup folder. (default: 5)
Filename Pattern	<input type="text"/> \${gatewayName}_Ignition-backup-\${e} The filename pattern used for creating scheduled and manually downloaded gateway backups. (default: \${gatewayName}_Ignition-backup-\${edition}\${timestamp}.gwbk)

Ideally, the folder you are saving scheduled backups into will be on a different computer than the one running Ignition and will be a folder that your IT department includes in their regular system backup cycle.

5. Specify a **Backup Schedule**.

The system uses CRONTAB scheduling, which is an offset from midnight. The order of the numbers is, from left to right, minutes, hours, days, months, day of week. Our on line documentation has details on working with CRONTAB at <https://docs.inductiveautomation.com/docs/8.1/appendix/reference-pages/crontab-formatting-reference>.

6. Specify the **Retention Count**.

This setting allows you to set the number of backups to keep in the folder. When this number is met, the system will automatically delete the oldest backup.

7. Specify the **Filename Pattern**.

The default pattern is the name of the Gateway, an underscore, the words “Ignition-backup,” the current version of this Ignition installation, and a timestamp.

8. Click **Save Changes**.

Project Backup and Restore

You can also backup your Projects individually. When doing this, remember that Gateway scoped resources are not part of the project. These backups do not include Tags, device connections, database connections, user sources, or anything else you have changed in the Gateway web page.

Project backups can be imported into any Gateway and are an effective way of loading a project into a new Gateway.

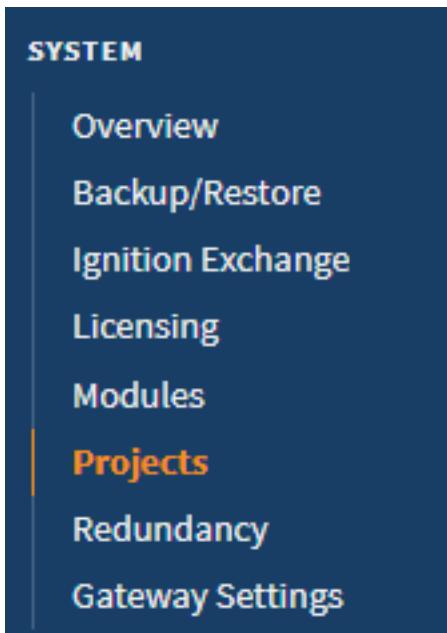
Remember that all projects are backed up with the Gateway is backed up, so this is not a required step for merely maintaining backups of projects. Instead, it should be seen as a way to move projects or specific project resources between Gateways.

Backup from the Gateway

You can back up an entire project from the Gateway. This option is quick and easy but does not allow you to specify which parts of the project are backed up, a filename, or a backup location.

Projects can be downloaded from the Gateway by anyone with access to the Gateway, and do not necessarily require Designer access.

1. Click **Projects** in the **System** section.



2. Click **More**.
3. Click **Export**.

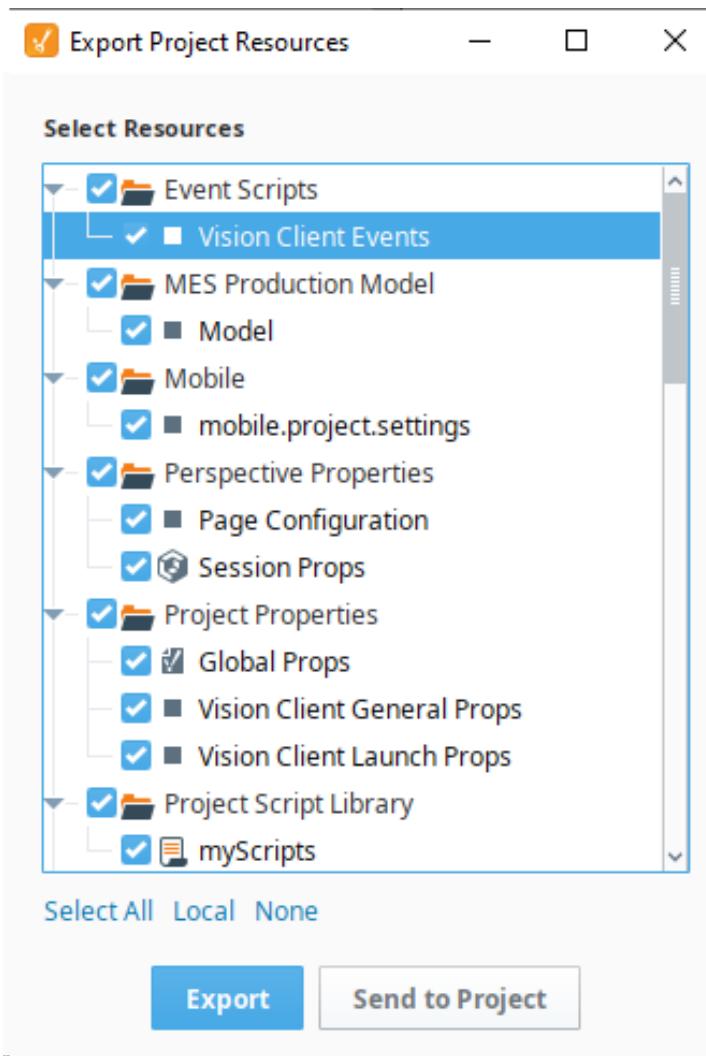
Your project will be downloaded to the browser's download location as a ZIP file.

Note: Even though the project file downloads as a regular ZIP file, it is not recommended that you unzip the file or alter any of the files and folders included in it. Doing so can damage the backup.

Backup from Designer

You can also make a project backup from the Designer. This gives you considerably more control of the backup. You can select the filename and location, and decide which resources within the Project you want to backup.

1. Return to **Designer**.
2. Click **File**.
3. Click **Export**.
4. Select the resources you wish to include in the backup.



Anything you have changed in the project will be available as a selection. You can use the All and None links at the bottom to quickly select or deselect every resource.

5. Click **Export**.
6. Select a **save location**.
7. If desired, enter a new filename.
8. Click **Export**.

As with the backup from the Gateway, do not unzip this file or change its contents.

Import a Project Backup

Importing from the Designer uses the same resource selection window as the export window. The resources from the imported project will be merged into the existing project.

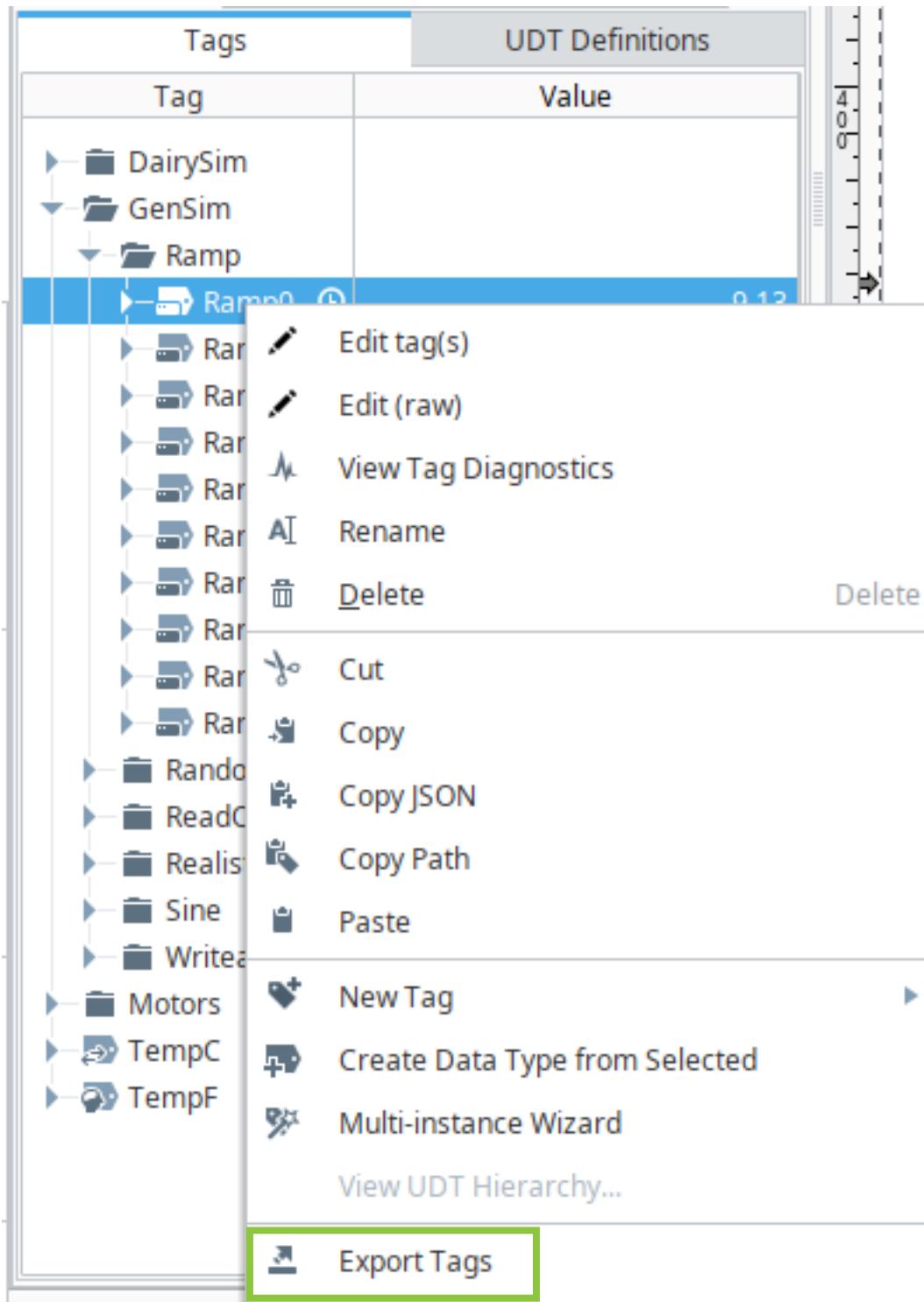
Backup Tags

Tags are backed up as part of a Gateway backup, but you can also back them up separately from the Designer. Note that Tag backups only contain the information necessary to import the Tag back into Designer.

Backup an Individual Tag

You can backup an individual Tag. Note that if the Tag in question is an instance of a UDT, you will also need to export and then import the UDT definition.

1. Expand **GenSim** in the Tag Browser.
2. Expand **Ramps**.
3. Right-click **Ramp0**.
4. Click **Export Tags**.



5. Select a save location.
6. Enter **ramp0.json** in the File name field.

Note: All Tag exports default to the same name—Tags.json. You will need to rename files as you download them if you are going to be doing multiple exports.

7. Click **Save**.

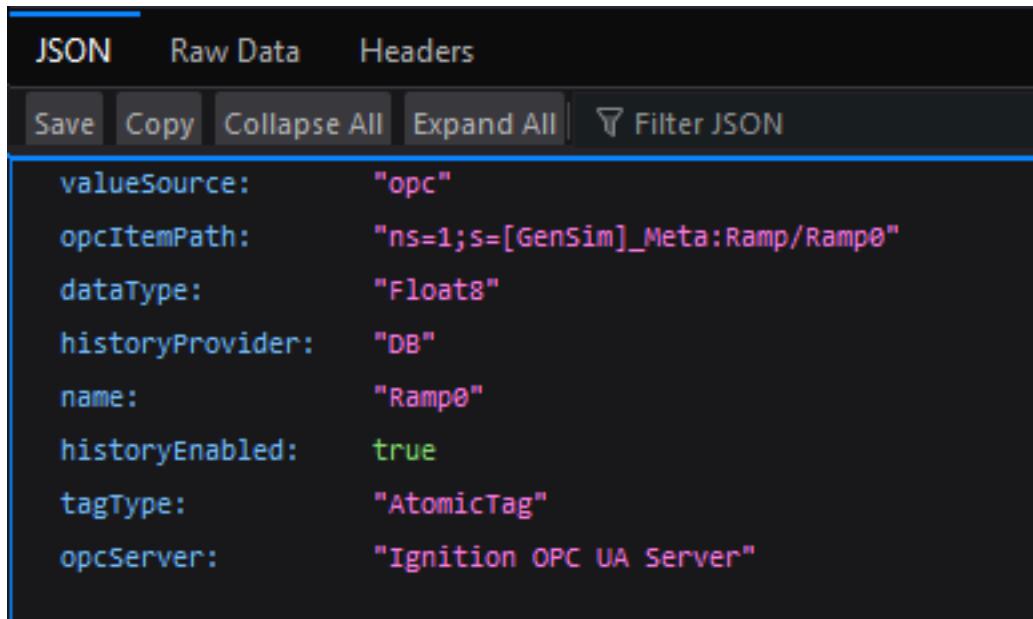
Examine a Tag Export

Tags export as a JSON file. JSON is a file format for describing data in plain text.

1. Open **Firefox**.

Note: If you do not have Firefox on your computer, you can download it for free from mozilla.org. Any modern web browser can open a JSON file, so the following steps will work in other browsers, but Firefox does a better job of formatting the file than most other browsers.

2. Press **Ctrl+O** on your keyboard.
3. Navigate to the file location where you saved the Tag export file.
4. Double-click the file.



The screenshot shows a JSON viewer with the following interface elements:

- Top navigation bar: JSON, Raw Data, Headers.
- Toolbar: Save, Copy, Collapse All, Expand All, Filter JSON.
- JSON content area:

```

    valueSource: "opc"
    opcItemPath: "ns=1;s=[GenSim]_Meta:Ramp/Ramp0"
    dataType: "Float8"
    historyProvider: "DB"
    name: "Ramp0"
    historyEnabled: true
    tagType: "AtomicTag"
    opcServer: "Ignition OPC UA Server"
  
```

You will see that the JSON file contains a set of definitions that provide all of the information needed to import the Tag into Designer.

Import a Tag

In order to demonstrate importing a Tag into Designer, we are going to delete Ramp0, then import it back into Designer using our backup.

1. Right-click **Ramp0**.

2. Click **Delete**.
3. Click **Yes**.
4. Right-click **Ramp**.
5. Select **Import Tags**.
6. Select **Direct**.

Direct import simply imports the Tag into the selected folder. Interactive import displays the same dialog box for importing Tags that we used when we first brought Tags into Designer. This give you more control when importing multiple Tags.

7. Navigate to the folder you selected when you exported the Tag.
8. Double-click **ramp0.json**.

The Tag is imported back into Designer.

Export Tag Folders and UDTs

In addition to exporting individual Tags, you can export an entire folder of Tags, and you can export UDTs. The process for either is the same as for an individual Tag—it's simply a matter of what you right-click on in the Tag Browser. If you do not have any Tags selected when you export, all Tags will be exported.

Importing folders and UDTs is also the same process as importing an individual Tag, but you will likely find the Interactive option more useful.